conference

proceedings

Proceedings of IMC 2005: Internet Measurement Conference 2005

IMC 2005 Internet Measurement Conference 2005

Berkeley, CA, USA

October 19–21, 2005

Sponsored by

ACM SIGCOMM

in cooperation with

The USENIX Association

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
URL: http://www.usenix.org

The price is $40.
Outside the U.S.A. and Canada, please add
$10 per copy for postage (via air printed matter).

ISBN 1-931971-37-4

Proceedings of

IMC 2005

Internet Measurement Conference 2005

Sponsored by ACM SIGCOMM

in cooperation with

The USENIX Association

October 19–21, 2005
Berkeley, CA, USA

Conference Organizers

Program Co-Chairs
Venkata N. Padmanabhan, Microsoft Research, USA
Darryl Veitch, University of Melbourne, Australia

Program Committee
Sharad Agarwal, Microsoft Research, USA
Jussara Almeida, Federal University of Minas Gerais, Brazil
Chadi Barakat, INRIA, France
Paul Barford, University of Wisconsin, USA
Supratik Bhattacharyya, Sprint Labs, USA
Mark Claypool, Worcester Polytechnic Institute, USA
Constantinos Dovrolis, Georgia Institute of Technology, USA
Steven Gribble, University of Washington, USA
Timothy Griffin, University of Cambridge, UK
Jasleen Kaur, University of North Carolina, Chapel Hill, USA
Balachander Krishnamurthy, AT&T Labs—Research, USA
Craig Labovitz, Arbor Networks, USA
Bruce Maggs, Carnegie Mellon University and Akamai, USA
David Moore, CAIDA and University of California at San Diego, USA
Dina Papagiannaki, Intel Research, UK
Rajeev Rastogi, Bell Labs, India
Jennifer Rexford, Princeton University, USA
Keith Ross, Polytechnic University, USA
Matthew Roughan, University of Adelaide, Australia
Neil Spring, University of Maryland, USA
Carey Williamson, University of Calgary, Canada
Walter Willinger, AT&T Labs—Research, USA
Lixia Zhang, University of California at Los Angeles, USA
Yin Zhang, University of Texas at Austin, USA

Local Arrangements Chair
Albert Greenberg, AT&T Labs—Research, USA

Steering Committee
Mark Crovella, Boston University, USA
Balachander Krishnamurthy, AT&T Labs—Research, USA
Bruce Maggs, Carnegie Mellon University and Akamai, USA
Nina Taft, Intel Research Berkeley, USA

The USENIX Association Staff

External Reviewers
Virgilio Almeida, Federal University of Minas Gerais, Brazil
Rob Austein, Internet Systems Consortium, USA
Steve Bellovin, Columbia University, USA
Amogh Dhamdhere, Georgia Institute of Technology, USA
Ruomei Gao, Georgia Institute of Technology, USA
Dorgival Guedes, Federal University of Minas Gerais, Brazil
Qi He, Georgia Institute of Technology, USA
Felix Hernadez-Campos, University of North Carolina, Chapel Hill, USA
Manish Jain, Georgia Institute of Technology, USA
Sharad Jaiswal, University of Massachusetts, Amherst, USA
Kevin Jeffay, University of North Carolina, Chapel Hill, USA
Ritesh Kumar, University of North Carolina, Chapel Hill, USA
Long Le, University of North Carolina, Chapel Hill, USA
Mark Lindsey, Engineers' Consulting Group, Georgia, USA
Nikitas Liogkas, University of California at Los Angeles, USA
Sridhar Machiraju, Sprint ATL, USA
Andre Madeira, Rutgers University, USA
Ravi S. Prasad, Georgia Institute of Technology, USA
Jeyashankher Ramamirtham, Bell Labs Research, India
Sushant Rewaskar, University of North Carolina, Chapel Hill, USA
Vivek Sawant, University of North Carolina, Chapel Hill, USA
Alok Shriram, University of North Carolina, Chapel Hill, USA
F. Donelson Smith, University of North Carolina, Chapel Hill, USA
Joel Sommers, University of Wisconsin, USA
Muhammad Mukarram Bin Tariq, Georgia Institute of Technology, USA
Nina Taft, Intel Research, Berkeley, USA
Jeff Terrell, University of North Carolina, Chapel Hill, USA
Alec Wolman, Microsoft Research, USA
Tao Ye, Sprint ATL, USA
Hui Zang, Sprint ATL, USA

IMC 2005:
Internet Measurement Conference 2005
October 19–21, 2005
New Orleans, LA, USA

Index of Authors

Message from the IMC 2005 Program Co-Chairs

Welcome to IMC 2005, the fifth in a series of highly successful conferences (and, previously, workshops) focused on Internet measurements. We are delighted to present an exceptionally strong program spanning the traditional areas of focus for IMC as well as new areas such as security and wireless networking. Of 148 papers submitted, 36 were accepted (22 long papers and 14 short papers) after a rigorous review process comprising PC member reviews, email discussion, and a full-day PC meeting. A total of 456 reviews were generated in the process, almost all by the PC members themselves. External reviewers provided input in some cases and full reviews in a few instances.

The strong program and conference are the result of hard work by many people: the authors, the PC members (the majority of whom were serving on the IMC PC for the first time), the external reviewers, John Papandriopoulos who helped run the CyberChair paper submission and review system, Casey Henderson and Jane-Ellen Long of USENIX who managed the conference Web site and the production of the printed proceedings, Mark Crovella who hosted the PC meeting at Boston University, Albert Greenberg who served as the local arrangements chair, and the IMC steering committee members (Mark Crovella, Balachander Krishnamurthy, Bruce Maggs, and Nina Taft) who were a constant source of advice. We thank them all. We also thank Intel Corp., Cisco Systems, Microsoft Research, USENIX, and ACM SIGCOMM for their generous support of the conference.

Venkat Padmanabhan and Darryl Veitch
IMC 2005 Program Co-Chairs

Measurement-based Characterization of a Collection of On-line Games

Chris Chambers Wu-chang Feng
Portland State University
{*chambers,wuchang*}@*cs.pdx.edu*

Sambit Sahu Debanjan Saha
IBM Research
{*ssahu,dsaha*}@*us.ibm.com*

Abstract

On-line games are a rapidly growing Internet application. Because of the cost in supporting on-line games and the unpredictable load on servers, companies are moving toward sharing infrastructure for game hosting. To efficiently provision on-line games, it is important to understand game workloads and the behavior of game players. In this paper, we present a comprehensive analysis of a collection of on-line game players and game workloads using data from several sources including: a 13-month trace of an extremely busy game server containing over 2.8 million connections, a two-year trace of the aggregate game populations of over 550 on-line games, and a 4-month trace of a content-distribution network used to deliver games. The key findings from our measurement study are: (1) these gamers are an extremely difficult set of users to satisfy and unless game servers are properly set up and provisioned, gamers quickly choose to go elsewhere, (2) the popularity of these games follows a power law making games difficult to provision at launch time, (3) game workloads are predictable only over short-term intervals, (4) there are significant challenges in hosting games on shared infrastructure due to temporal and geographic synchronization across different games and other interactive applications, and (5) game software updates are a significent burden on game hosting that must be planned for. Our results have implications for both game publishers as well as infrastructure providers.

1 Introduction

On-line gaming is an increasingly popular form of entertainment on the Internet, with the on-line market predicted to be worth over $5 billion dollars in 2008 [1]. As an example of a popular, money-making game, EverQuest [2] has over 450,000 subscribers each paying a monthly fee and purchasing two yearly expansions. Unfortunately for game companies, the success of a game is highly unpredictable. To make matters worse, there are substantial costs in developing and hosting on-line games. As a result, such companies are increasingly exploring shared, on-line hosting platforms such as on-demand computing infrastructure provided by companies such as IBM and HP [3, 4, 5, 6, 7, 8, 9, 10].

In order to judge the feasibility of such an approach, it is important for game and hosting companies to understand how gamers and game workloads behave. Knowing the behavior of players, the predictability of workloads, and the potential for resource sharing between applications allows infrastructure to be tailored to the needs of games. While there has been a substantial amount of work characterizing web and peer-to-peer users and workloads [11, 12], there is very little known about game players and workloads.

In order to provide insight into such issues, this paper examines several large traces of aggregate player populations of a collection of popular games as well as the individual player population of a busy game server. We present a detailed analysis of on-line game players and workloads that targets several key areas which are important to game and hosting providers including:

- *How easy is it to satisfy gamers?*: One of the key issues in providing a successful game is to understand how players connect to servers and how long they play on them. By understanding what players are willing to put up with, game and hosting companies can tailor their infrastructure and content to maximize player satisfaction. For example, one of the challenges with using on-demand computing infrastructure for games is the latency associated with re-purposing a server. It would thus be useful to characterize how patient game players are in connecting to a game before deploying such infrastructure. To this end, we characterize individual player behavior of an extremely popular Counter-Strike game server over a long period of time. Our results show that gamers are an extremely difficult set of users to satisfy and that unless game servers are properly set up and provisioned, gamers quickly choose to go elsewhere.

- *How predictable are game workloads?* Another problem in hosting on-line games is determining the amount of hardware and network bandwidth that is required. Hosting a game is an expensive proposition, costing the game provider more than 30% of the subscription fees in just hardware and bandwidth per month [13]. Hosting is made all the more difficult by variations of popularity as the game moves through its life cycle. Game companies face the provisioning problem both in determining the amount of resources to provide at launch time and in allocating spare resources to support dynamic usage spikes and subscriber growth. Characterizing the diversity and predictability of game workloads allows companies to more accurately provision resources. To this end, we examine the real-time aggregate game player population of more than 550 on-line games, the most popular of which are first-person shooters. Our results show that the popularity of these games follows a distinct power law distribution making the provisioning of resources at launch-time extremely difficult. However, as games mature, their aggregate populations do become predictable, allowing game and hosting companies to more easily allocate resources to meet demand.

- *Can infrastructure be shared amongst game and other interactive applications?* With the advent of commercial on-demand computing infrastructure, it is becoming possible to statistically multiplex server resources across a range of diverse applications, thus reducing the overall hardware costs required to run them. In order for such shared infrastructure to provide any savings, peak usage of applications must not coincide. To characterize the amount of sharing benefit that is available, we examine the usage behavior of a number of popular on-line games and compare them against each other and against the usage behavior of several large distributed web sites. As on-demand infrastructure is distributed, we also examine the client load of a number of servers based on geographic region. Our results show that usage behavior of interactive applications follows strict, geographically-determined, time-of-day patterns with limited opportunities for resource sharing.

Section 2 describes the methodology behind our study. Section 3 analyzes properties of individual gamers. Section 4 describes trends of on-line gaming in aggregate. Section 5 evaluates the potential for multiplexing games and web traffic together, and Section 6 discusses our conclusions.

`cs.mshmro.com` trace	
Start time	Tue Apr 1 2003
End time	Mon May 31 2004
Total connections	2,886,992
Total unique players	493,889

GameSpy trace	
Start time	Fri Nov 1 2002
End time	Fri Dec 31 2004
Total games	550
Total player time	337,765 years

Steam CDN trace	
Start time	Mon Sep 27 2004
End time	Mon Apr 8 2005
Content transferred	6,193 TB
Average transfer rate	3.14 Gbs

Table 1: Data sets

2 Methodology

The study of on-line game usage is typically limited due to the proprietary nature of the industry. To overcome this, we have collected several unique data sets that allow us to analyze properties that have not been possible previously. These data sets include the following:

Individual player data: In order to study the behavior of individual players playing a representative on-line game, we examined the activity of one of the busiest and longest running Counter-Strike servers in the country located at `cs.mshmro.com` [14, 15]. Counter-Strike (a Half-Life modification) is currently the dominant on-line game with the largest service footprint of any game at 35,000 servers and over 4.5 billion player minutes per month [16]. Of all of the active Counter-Strike servers, `cs.mshmro.com` is among the busiest 20 servers as ranked by ServerSpy [17]. The server averages more than 40,000 connections per week, has hosted more than 400,000 unique players within the last year, and has logged more than 60 player years in activity since its launch in August 2001. Table 1 describes the trace collected from the server.

GameSpy aggregate player population data: One problem with measuring on-line game usage is the limited access to game server hosting data. Game companies typically keep the access and usage behavior of their players confidential. There are two factors that enable the measurement of aggregate game player populations, however:

(1) on-line games use a centralized authentication server to keep track of the players that are playing and (2) information on overall player numbers per game is usually exported publicly. Several game portal services collect such player numbers over a large number of games and report the information in real-time. Among these services is the GameSpy network, which provides real-time player population data on individual games in a structured format that can readily collected and analyzed [18]. Currently, there are over 550 on-line games that are being tracked across various genres including first-person shooter games (FPS), massively multi-player on-line role-playing games (MMORPG), real-time strategy games (RTS), card and board games, and sports games. The most popular games tracked by the Gamespy network are from the FPS genre however, and therefore when we refer to gamers we are predominately referring to FPS gamers. To study on-line game population behavior, we have collected a data feed from GameSpy for more than two years since November 2002. Our redundant collection facility periodically samples the GameSpy data every 10 minutes. Note that the availability of the data is sensitive to many factors, including service outages at the portal and our own outages. These outages have been manually removed from the data analysis. Table 1 describes the data set which includes over 50 million measurements and represents more than 300,000 years of player time spent on games over the course of the last two years.

Content-distribution network: One of the common features of on-line games is their ability to dynamically update themselves. To support this feature, many games employ custom, game-specific, content distribution networks that deliver new game content and software patches to clients when needed. One such network is Steam [19], a multi-purpose, content-distribution network run by Valve which is used to distribute run-time security modules as well as client and server software patches for Half-Life and its mods such as Counter-Strike and Day of Defeat. The network consistently delivers several Gbps of content spread across over 100 servers. In order to analyze the resource usage of Steam, we have collected its data feed over the last 6 months, a duration that has seen Steam deliver more than 6 petabytes of data. Table 1 describes the trace collected.

3 Gamers as individuals

It is important for game providers to understand the usage behavior of its players in order to adequately address their needs. In order to study player characteristics, we analyze the trace of `cs.mshmro.com` to track individual gamers throughout their play cycle. Specifically, we track gamers attempting to connect to the server, gamers playing on the server, and the likelihood of a gamer returning to the server.

We first demonstrate that gamers are difficult to please.

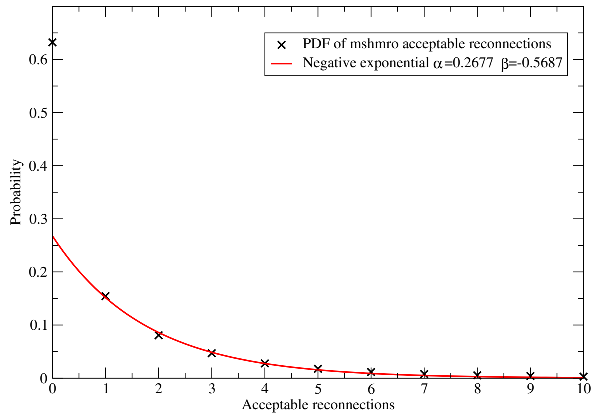


Figure 1: PDF of player impatience based on number of acceptable reconnects

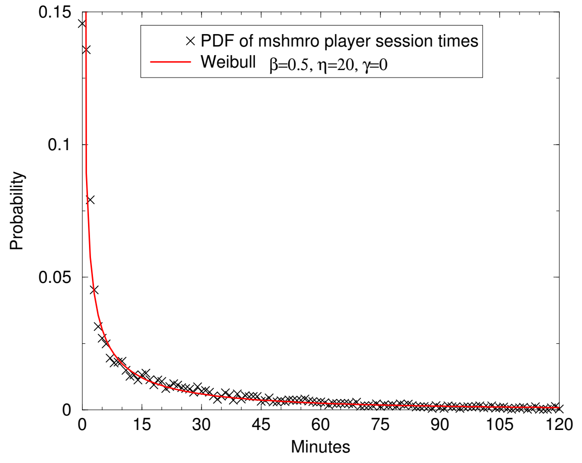
In particular, they 1) have no tolerance for busy servers, often connecting once while the server is busy and never reconnecting again for the entire trace, 2) have very specific gameplay needs and if those needs are not met in the first few minutes of play, their likelihood of continuing to play at the server drops off dramatically, and 3) they often have no loyalty or sense of community tied to a specific server and do not return after playing a handful of times. For those that do return often, we also demonstrate that their session times show a marked decline and their session interarrival times show a marked increase just as they are ready to quit playing on the server altogether.

3.1 Gamers are impatient when connecting

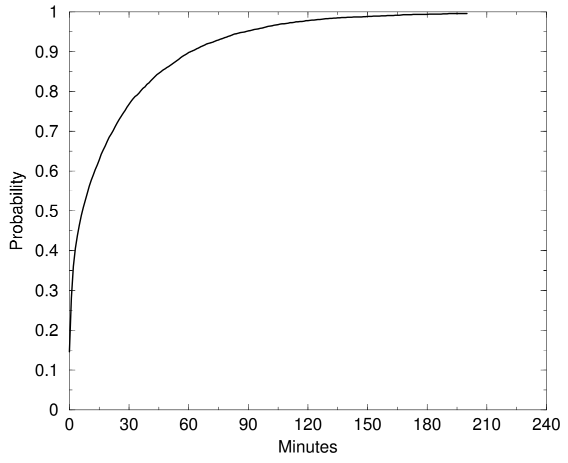
Quantifying the patience of on-line gamers is important for adequate server provisioning. For some Internet applications, such as web-browsing, users are known to be impatient [20]. For others, such as peer-to-peer services such as *Kazaa*, users are very patient [12].

Our trace of `cs.mshmro.com` records successful connections as well as connection attempts, when players connect to the server and are refused service. The latter is extremely common; every day, the server turns away thousands of people. Browsing the trace, it is not unusual to see the same player reconnect to the server several times in a row, waiting for a spot on the server to free up. We operate on the assumption that a player's willingness to reconnect to the same busy server repeatedly is an indication of their patience.

In order to quantify player patience we group each player's connection history into sessions, and consider a session of length N evidence of that player's willingness to reconnect after $N-1$ connections. Figure 1 shows the probability distribution of acceptable reconnections per



(a) PDF



(b) CDF

Figure 2: Session time results for `cs.mshmro.com` trace

player. As the figure shows 73% of the players are unwilling to reconnect to the server enough to play even once. One of the reasons players do not reconnect is that game clients have a "Quick Start" mechanism that many players use. The mechanism works by downloading a list of candidate servers from the master server and cycling through them one by one until a successful session is established. Thus, such clients may not lack patience, but rather are automatically redirected elsewhere. For the rest of the players, however, 13% are willing to reconnect one time on average with the percentage sharply decreasing on successive reconnects. Aside from the first data point, the rest of the graph represents a client's patience in connecting to our busy server and, not surprisingly, can be fit very closely with a negative exponential distribution. As Figure 1 shows, a negative exponential distribution with parameters $\alpha = 0.2677$ and $\beta = -0.5687$ fits the data with a correlation coefficient of 0.998. Players, therefore, exhibit a remarkable degree of impatience with busy game servers.

3.2 Gamers have short attention spans

Using the same trace, we extracted the total session time of each player session contained in the trace. Figure 2 plots the session time distributions of the trace in unit increments of a minute [1]. The figure shows, quite surprisingly, that a significant number of players play only for a short time before disconnecting and that the number of players that play for longer periods of time drops sharply as time increases. Note that in contrast to heavy-tailed distributions reported for most source models for Internet traffic; the session ON times for game players is not heavy-tailed. To further illustrate this, Figure 2(b) shows the cumulative density function for the session times of the trace. As the figure shows, more than 99% of all sessions last less than 2 hours.

[1]Note that a preliminary version of our results here were first reported in a short paper at the NetGames 2003 Workshop [21]

Unlike the player patience data, session times can not be fitted with a simple negative exponential distribution. However, the data can be closely matched to a Weibull distribution, a more general distribution that is often used to model lifetime distributions in reliability engineering [22]. Since quitting the game can be viewed as an attention "failure" on the part of the player, the Weibull distribution is well-suited for this application. The generalized Weibull distribution has three parameters β, η, and γ and is shown below.

$$f(T) = \frac{\beta}{\eta}(\frac{T-\gamma}{\eta})^{\beta-1}e^{-(\frac{T-\gamma}{\eta})^{\beta}}$$

In this form, β is a shape parameter or slope of the distribution, η is a scale parameter, and γ is a location parameter. As the location of the distribution is at the origin, γ is set to zero, giving us the two-parameter form for the Weibull PDF.

$$f(T) = \frac{\beta}{\eta}(\frac{T}{\eta})^{\beta-1}e^{-(\frac{T}{\eta})^{\beta}}$$

Using a probability plotting method [22], we estimated the shape (β) and scale (η) parameters of the session time PDF. As Figure 2(a) shows, a Weibull distribution with $\beta = 0.5$, $\eta = 20$, and $\gamma = 0$ closely fits the PDF of measured session times for the trace.

This result is in contrast to previous studies that have fitted a negative exponential distribution to session-times of multiplayer games [23]. Unlike the Weibull distribution which has independent scale and shape parameters, the

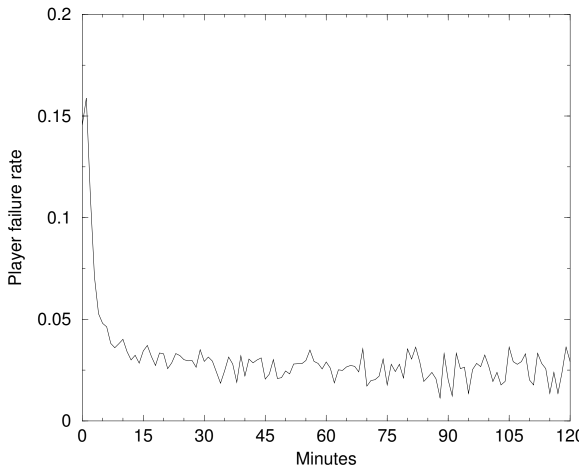


Figure 3: Player failure rates for individual session times for `cs.mshmro.com` trace

shape of the negative exponential distribution is completely determined by λ, the failure rate. Due to the memory-less property of the negative exponential distribution, this rate is assumed to be constant. Figure 3 shows the failure rate for individual session durations over the trace. As the figure shows, the failure rate is *higher* for flows of shorter duration, thus making it difficult to accurately fit it to a negative exponential distribution. While it is difficult to pinpoint the exact reason for this, it could be attributed to the fact that Counter-Strike servers are notoriously heterogeneous. Counter-Strike happens to be one of the most heavily modified on-line games with support for a myriad of add-on features [24, 25]. Short flows could correspond to players browsing the server's features, a characteristic not predominantly found in other games. As with player patience, it may be possible to fit a negative exponential for longer session times. As part of future work, we hope examine this as well as characterize session duration distributions across a larger cross-section of games to see how distributions vary between games and game genres.

3.3 Gamers are not loyal

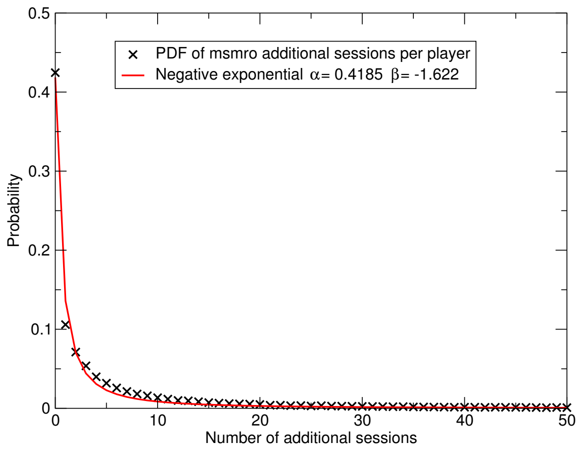
Public-server games such as Half-life provide users with a large choice of servers located all around the world. Gamers can switch between servers as often as they like. Some reasons to continue playing on the same server are simplicity, a known low-latency connection, preference for server options, or a sense of community. It is natural to wonder whether servers continue to serve the same group of clients and to what extent these reasons or others keep clients at a specific server.

Our trace contains the connection records for each client via their unique player identification number (WONID). We quantify loyalty to the server by counting the number of times a player returns to play after a successful playing session. Figure 4(a) shows the probability density function of additional game sessions per player for players who returned at least once to the server while Figure 4(b) shows, on a logarithmic scale, the cumulative distribution. As the figure shows, 42% of the players in our trace returned to play only once and 81% played less than 10 times. On the other hand, the top 1% of loyal gamers return to play hundreds of times (hence the logarithmic scale). It appears that the majority of clients have very little loyalty to public servers, and only a small fraction have grown strongly attached. We hypothesize that, due to a large population of servers to choose from (over 30,000), clients rarely select the same server twice.

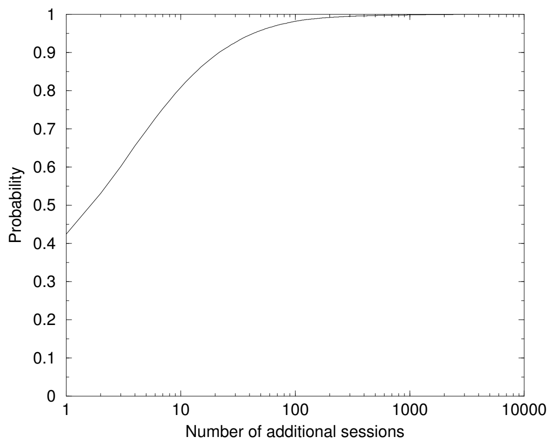
3.4 Gamers reveal when they lose interest

Players of a game have some discretion about how frequently they play a game and for how long. Players often lose interest in a game and cease playing altogether at some point. Before that happens, however, there may be noticeable indications that their interest is waning. Such indications are extremely useful to game providers who can detect waning interest and react to it on a macro level with new content or on a per-player basis via customized incentives for continued play.

We determine the average player interest curve by calculating each player's sequence of play sessions from their first session to their last recorded session. This is a player's *play history*. Since each player may progress through his or her game interest at a different rate, we normalize each of these data sets based on the duration each player is active on the server. We then examine the average session times and session interarrival times of all players throughout their playing careers. Figure 5(a) shows that player session times are relatively constant halfway through their play history and fall off to just more than 50% of the initial session time before the player loses interest completely. Figure 5(b) shows that the time between player sessions is minimized before the halfway point and increases steeply until the player's interest has fully waned. The variance on this data is extremely high, due in part to the fact that players only spend a portion of their time on this single server, and therefore this data is unsuitable for predicting the interest of a given player. However we believe this methodology can be used for games with a centralized session-tracking authority as an early indicator of peaking player interest and that game publishers should use these measurements to trigger the delivery of new content or incentives for the individual player.

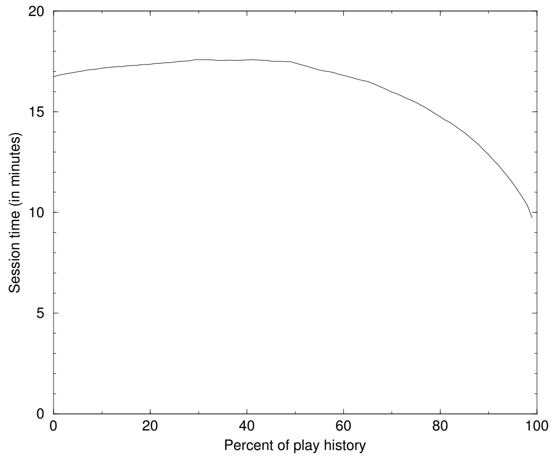


(a) PDF

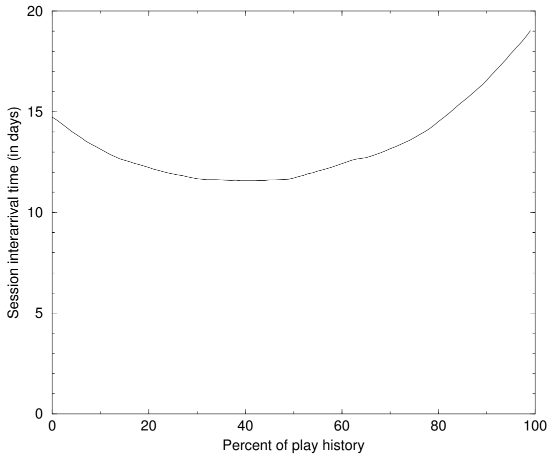


(b) CDF

Figure 4: Distribution of sessions per player on the server



(a) Average session time



(b) Average session interarrivals

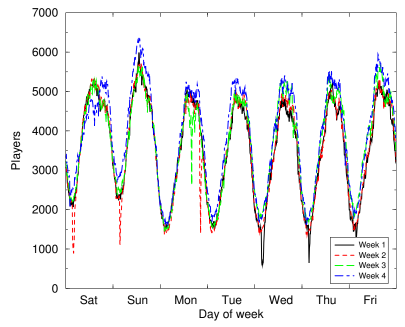
Figure 5: Player behavior throughout their playing careers

4 Game populations

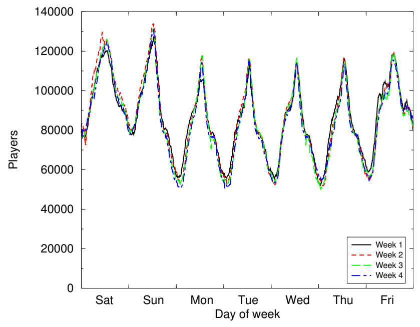
Hosting games is challenging, in part, due to the difficulties of accurate provisioning. Under-provisioning can test gamer patience, while over-provisioning can be costly. We look at two facets of gaming integral to successful game provisioning: overall game popularity and predicting game workloads. We show that (1) there are, and will be, very few extremely popular games, and (2) game workloads are periodic and predictable over short-term intervals.

4.1 Game popularity follows a power-law

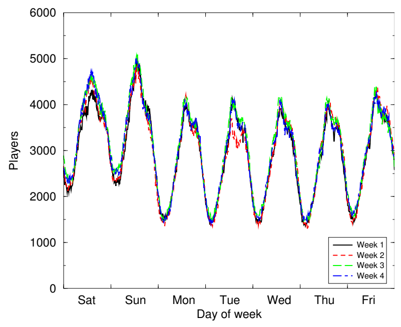
To determine the distribution of on-line game popularity, we analyzed a nine-month subset of the GameSpy data set described in Section 2, starting March 1st 2003. Of the games, we consider only the top 50 games, as the remaining games averaged a minimal number of players throughout the trace. To average popularity rankings we first calculated the rank ordering of the games and the number of players at a given rank for each day. Then we averaged these daily rankings over the nine-month period to show the distribution of players across the games regardless of fluctuations in individual game popularity. Figure 6 shows the popularity data on a log-log scale. As the figure shows, this distribution is heavily skewed in favor of the most popular games, with the first ranked game having over ten times the number of players of the next most popular. This distribution of popularity is most similar to a power-law distribution. Power-law distributions are of the form $y = ax^{\lambda}$



(a) America's Army



(b) Half-Life



(c) Neverwinter Nights

Figure 7: Player load for three popular games over a 4-week period

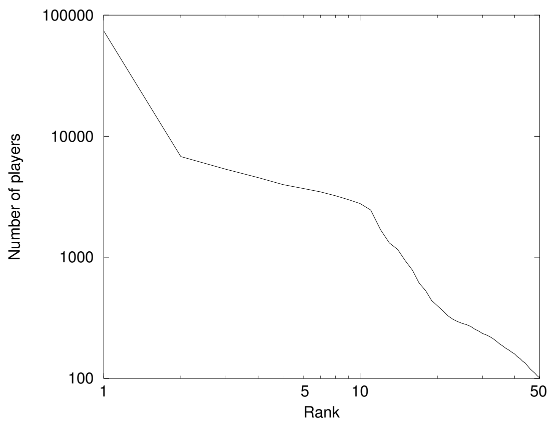


Figure 6: Game popularity distribution averaged over nine months (log scale)

and occur in a number of places including the frequency of words in the English language, the popularity of web pages, and the population of cities. An intuition for these distributions is that whenever choices are made between many options, and each choice affects other choices, the choices tend to pile up on a few popular selections. Games and servers create communities: in selecting one, each player's choice affects and is affected by the popularity and reputation of that game or server. A perfect power-law distribution would graph as a straight line on a logarithmic scale in both the x and y axis. The relatively straight line (correlation coefficient -0.98 for a simple linear regression) demonstrates that the GameSpy data does follow a power law distribution. This distribution has an interesting, albeit unfortunate, implication for provisioning server resources for on-line games: the host must plan for several orders of magnitude of change in popularity (and therefore resources) in either direction. As a result, this indicates that on-demand infrastructure can significantly reduce the costs and risks of launching and hosting on-line games.

4.2 Game workloads have varying degrees of predictability

Accurately predicting game workloads allows game hosting providers to allocate the appropriate amount of resources for a game. In order to determine whether this is feasible, we analyze the GameSpy trace for different sets of games. Specifically, we investigate whether any simple trends or patterns can be used to accurately predict the game workload, whether the workload is stable and if so, over what time scale.

4.2.1 Game workloads exhibit predictable daily and weekly changes

Intuitively, it is reasonable to assume that usage is strongly tied to the daily and weekly activities of players. Figure 7 shows the global player population of four consecutive weeks starting from 3/1/2003 for three popular games: America's Army, Half-Life, and Neverwinter Nights. As expected, the figure shows that the workload has regular daily cycles and that over this one month period the workload does not vary significantly from week-to-week. In fact, for all three games, the trends as well as the maximum and minimum points match up at identical points in time during the week. We observe similar results over other parts of the year with the only anomalies caused by service outages and by holidays. To further demonstrate the cyclical nature of gaming workloads, we take one year's worth of game server load samples across a variety of games and plot the Fast Fourier Transform (FFT) of the data. The FFTs have been scaled so that they can be plotted together. As Figure 8 shows, the FFT contains strong peaks at the 24-hour cycle for each of the games. There is also a significant peak at the 168-hour (one week) cycle for two of the games as well. This corresponds to an increase in player usage on

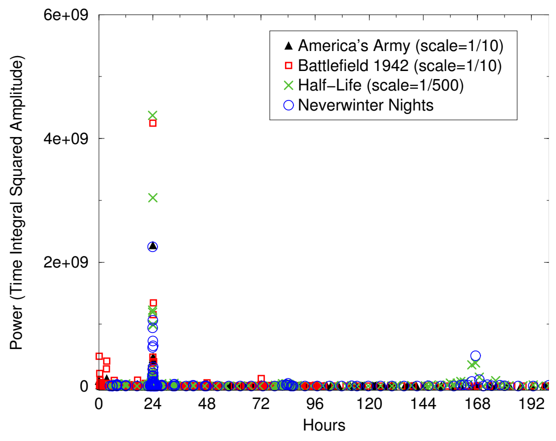


Figure 8: FFT of the player load from four games over one year.

the weekends during some parts of the year. Papagiannaki et. al use wavelet multiresolution analysis (MRA) on another long-term data series [26], and model their series as a 12-hour and 24-hour cycle plus a trend. We were unable to apply this technique however, due to the reliance of wavelet MRA on resolutions that are factors of two apart. The difference between our two cycles is seven.

In order to quantify the week-to-week variation of game workloads, Figure 9 shows distribution of week-to-week load changes of the top 5 games during 2004: Half-Life, Battlefield 1942, Medal of Honor: Allied Assault, America's Army, and Neverwinter Nights. Figure 9(a) plots the distribution of instantaneous load changes between identical points in time of consecutive weeks, while Figure 9(b) plots the change in average daily load between the same day of the week of consecutive weeks. Finally, Figure 9(c) plots changes in maximum daily load between the same day of the week of consecutive weeks. The figures fit a 't' location-scale distribution, which has three parameters, a scale parameter $\sigma > 0$, a location parameter μ, and a shape parameter $\nu > 0$. The density function for this distribution is as follows:

$$f(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sigma\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(\frac{\nu+(\frac{x-\mu}{\sigma})^2}{\nu}\right)^{-\frac{v+1}{2}}$$

Note that if x is 't' location-scale distributed, $\frac{x-\mu}{\sigma}$ is Student's 't' distributed with ν degrees of freedom. As illustrated in Figure 9, we find a very good fit for all the three plots. Based on this observation, we draw two main conclusions with regard to resource usage:

- As the figures show, almost all week-to-week load variations are under 10% of the previous week's workload. Such behavior makes it relatively easy for game and infrastructure providers to provision and predict resource usage on a weekly basis.
- The above distribution fitting of load variations indicates that it is feasible to model the week-to-week load variations using such standard distributions. We are exploring the feasibility of online parameter estimations for using this model in resource provisioning.

4.2.2 Game workloads exhibit unpredictable long-term fluctuations

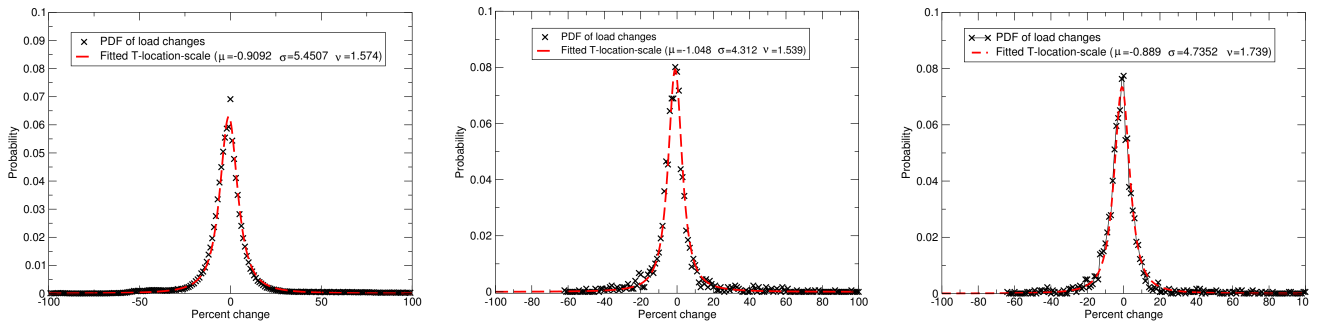
While the daily and weekly cycles in server load are clear, the duration of our trace allows us to examine longer term cycles. We examine the trend of three games of similar popularity as well as the trend of the most popular game, Half-life, over the period of just over two years. We compute the trend as the moving average of the data with a window size of one week. Figure 10 shows the trends of the respective games. The underlying trend of these games does not reveal periodicities on a monthly timescale, and the limits of our trace prevent us from drawing any strong conclusions about annual cycles. There are several points in trace where the games appear to be synchronized, but the explanation for the concurrent peaks or valleys is not necessarily predictable. We observe peaks in all games near the Christmas season, but, for example, all four games experience a drop during the unpredictable weeks of the *Sobig* virus [27].

5 Potential for multiplexing gain

With the movement toward hosted game services [28, 29] as well as on-demand computing infrastructure for games such as Butterfly.net [30], there has been a great deal of interest in reducing the cost of running game servers by sharing server resources dynamically across multiple games and applications. We explore two likely scenarios: hosting multiple games on the same servers, and hosting web sites along with game servers. In addition, we study the usage behavior of a content-distribution network for supporting games. Our results show that there are significant challenges in multiplexing interactive applications on the same server infrastructure and that only limited opportunities for reducing peak resource usage exist.

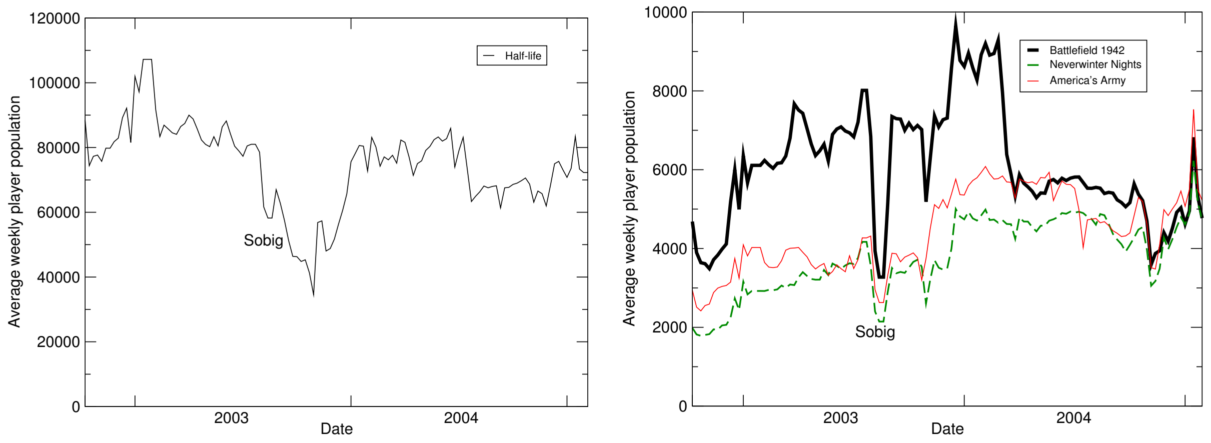
5.1 Game workloads are synchronized

There are two ways games can be multiplexed with each other. One way would be to coarsely and statically assign physical servers to particular games based on the popularity of the game. Results from Section 4 clearly show that this can provide a lot of benefit for game companies. Another



(a) Instantaneous (b) Mean (c) Maximum

Figure 9: Week-to-week PDF of percent load changes for the top 5 games of 2004



(a) Half-Life (b) Other games

Figure 10: Population trends for Half-life and other games with an averaging window of one week

Game	Average number of players
Half-Life	80324
America's Army	5791
Battlefield 1942	5402
Neverwinter Nights	4579

Table 2: Mean player populations for week of May 23, 2004

way would be to dynamically re-allocate servers based on instantaneous demand for a particular game. An implicit assumption that gives value to the latter method is that different games have usage patterns that are substantially different. Thus, rather than have each game provision server resources based on the peak usage of their game, server resources would be provisioned for the global peak.

In order to investigate the extent to which different games can be multiplexed with each other, we examined the aggregate player populations of four popular games. The games examined include FPS games (Half-Life, Battlefield 1942, and America's Army), as well as an MMORPG (Neverwinter Nights). Player populations of these games were collected over a one week period (Sunday May 23, 2004 to Saturday May 29, 2004) from the GameSpy trace. In order to compare the games directly, independent of their popularity, each game's population data was normalized by the mean population for that particular game during the week. Table 2 lists the mean player populations for the four games examined. Figure 11 plots the normalized player loads for the four games during the one week period. As the figure shows, player populations fluctuate significantly based on the time of day from lows close to half of the mean to peaks close to twice the mean. In addition, populations across games have peaks in close proximity to each other, making it difficult to achieve significant statistical multiplexing gain between different games. Finally, as indicated in the FFTs from Figure 8, games show

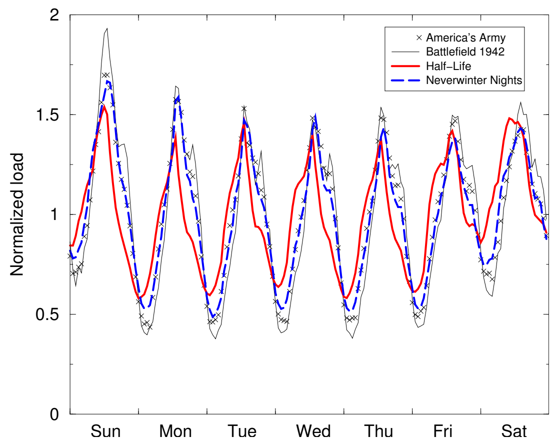


Figure 11: Aggregate normalized load across four popular games for week of May 23, 2004

slight peaks on the weekends with slightly more players on-line than during the week.

5.2 Games and interactive application workloads are synchronized

While Section 5.1 shows the difficulty in obtaining statistical multiplexing gain between different games, on-demand computing infrastructure could still be useful for multiplexing between other applications such as web servers. In order to examine this, we obtained web server logs over a week for three commercial sites. The sites included those for a North American cereal manufacturer, a North American credit card company, and an international beverage manufacturer. Table 3 describes the traces of the web servers, all from the week of August 13, 2001. The servers themselves were located in geographically distributed data centers and the individual logs from each site were aggregated and sorted into a single log file. Using these traces, we plotted the normalized load for the web server against the normalized global aggregate load of Half-Life during the same week in August 2004.

As Figure 12 shows, workloads for web and on-line games share similar daily periodic peaks. This particular week of game traffic does not have a strong weekend rise (perhaps due to being from the summer), but the web traffic does slump during the weekends as Figures 12(a) and 12(b) show. Interestingly, Half-life shows considerably less variance than the North American websites, but similar variance to the international beverage manufacturer website. Intuitively, it makes sense that applications and web sites with global usage patterns are more consistently busy and have less daily variance. Due to the international popularity of Half-Life, its usage pattern is quite similar to that of the international beverage company's web site. Overall, these results indicate that infrastructure sharing between applications during the week will have a somewhat limited benefit with some potential for multiplexing gain during the weekends and during the "off hours" for geocentric applications.

North American cereal manufacturer	
Start time	Mon Aug 13 2001
End time	Sun Aug 19 2001
Total requests	10,368,896
Content transferred	59.6 GB

North American credit card company	
Start time	Tue Aug 14 2001
End time	Mon Aug 20 2001
Total requests	112,590,195
Content transferred	366.4 GB

International beverage manufacturer	
Start time	Tue Aug 14 2001
End time	Sat Aug 18 2001
Total requests	11,932,946
Geographically resolvable	11,829,429
Content transferred	51.1 GB

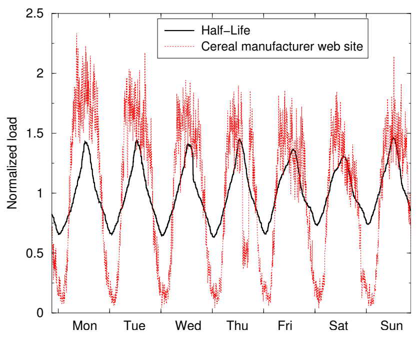
Table 3: Web site logs for week of August 13, 2001

Total connections	71,253
Geographically resolvable	30,226
From North America	9,414
From Asia	9,814
From Europe	8,788
From other continents	2,210

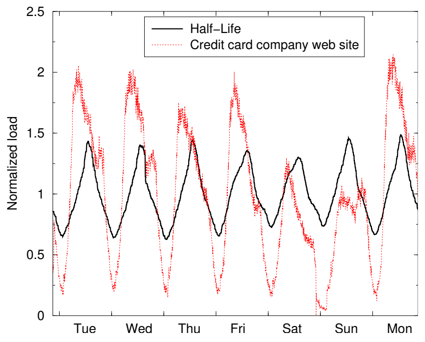
Table 4: Connection data for `cs.mshmro.com` for week of May 23, 2004

5.3 Games exhibit strong, diurnal geographic patterns

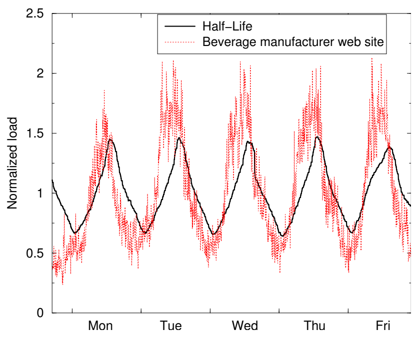
One of the salient features of globally distributed, on-demand computing infrastructure is that it can easily shift resources geographically close to where the demand is coming from. Intuitively, it makes sense that a predictable, diurnal pattern drives global resource consumption and hence, the provisioning of server resources. This is especially the case for applications that require human participants such as games. To study this phenomenon, we examined a one-week period of `cs.mshmro.com` (Sunday May 23, 2004 to Saturday May 29, 2004). Using this log and a commercial geographic IP address mapping tool [31], the location of each player connecting was resolved. As Ta-



(a) North American cereal manufacturer



(b) North American credit card company



(c) International beverage manufacturer

Figure 12: Aggregate normalized load between Half-Life and commercial web sites

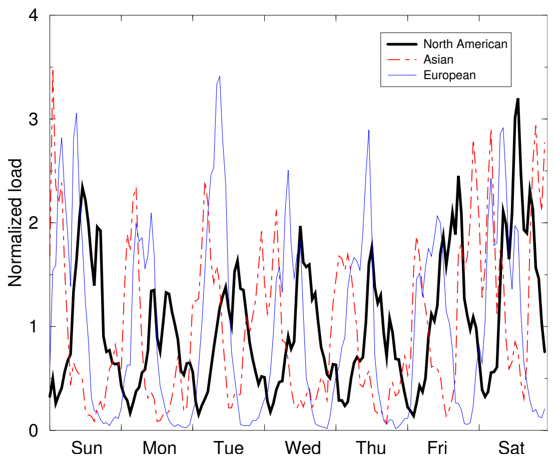


Figure 13: Aggregate normalized load per-continent for `cs.mshmro.com`

ble 4 shows, a significant portion of the load is from outside of North America. Using the resolved connections, the per-continent load normalized by the mean connection arrival rate was plotted. As Figure 13 shows, each continent shows a predictable, diurnal pattern of activity with the only difference being a time-zone shift. It is interesting to note that in contrast to the Half-Life aggregate load and international beverage company web site load (Figure 12(c)), the per-continent load of `cs.mshmro.com` exhibits a large variance similar to the North American web site loads shown in Figures 12(a) and 12(b). We hypothesize that when the usage patterns of international services are broken out into individual regions, the resulting load variances are similar to those of regional servers such as the cereal manufacturer and the credit card company.

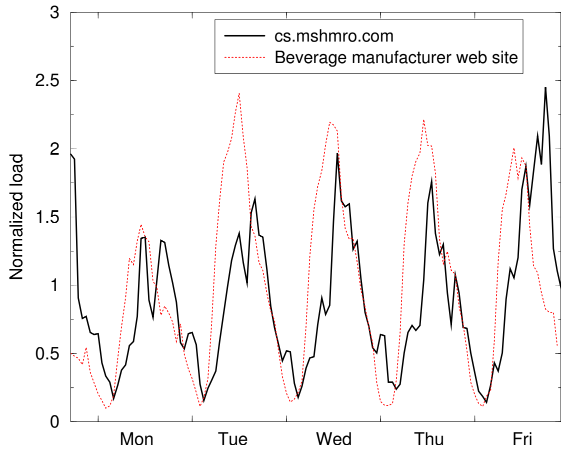
To test this hypothesis, we compared the per-continent load between `cs.mshmro.com` and the international beverage company web server trace [2]. Figure 14 shows the per-continent, normalized load of the game and web server for North America and Europe. The loads from other continents show similar results. As expected, the per-continent load fluctuations and variance are similar to those found in the two regional web sites. The figure also shows that usage of both applications are highly synchronized when broken down into geographic regions. The degree of synchronization thus limits the benefits that geographically distributed, on-demand computing infrastructure has on interactive applications such as games and web.

5.4 Game updates significantly impact resource usage

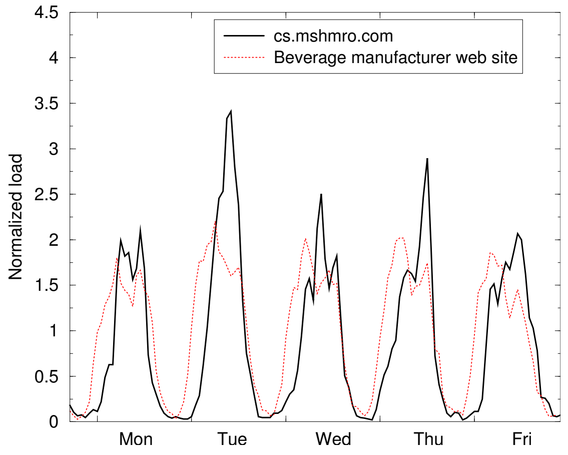
The infrastructure required to host on-line games must also account for the mutability of the games over time. Software patches to fix bugs, prevent cheats, and deliver new content to end-users are an expected component of many on-line games. These patches can vary greatly in size, from a few bytes to several gigabytes. Understanding the impact of these patches on hosting, and adequately provisioning for them is an important part of supporting on-line games. We use the trace of the Steam content delivery network to examine this aspect of games. Our Steam trace includes the initial download of the popular FPS game *Half-Life 2* as well as a number of sizable content updates for both clients and servers.

The Steam network is utilized for both player authentication and content distribution. Players are authenticated to Steam for each game session, via the download of an authentication module. Content is distributed to players (and

[2]Note that in comparing the geographical resolution data of Tables 3 and 4, a much larger percentage of the IP addresses in the beverage company trace is resolvable. This is due to the fact that the trace (and the set of IP addresses in it) is much older, giving services such as GeoBytes more time to identify their locations



(a) North America



(b) Europe

Figure 14: Normalized load for `cs.mshmro.com` and the international beverage company website

servers) via Steam at irregular intervals and irregular sizes. These two functions are not distinguished in the data set we have collected. However, we can differentiate them by utilizing the GameSpy dataset, which tracks player load, by assuming that player load and game authentication are linearly correlated.

As a way of validating that the Steam data and the GameSpy data are tracking the same thing (i.e. player load), we consider a week without a Steam update. Figure 15 shows a scatter plot of Steam data (in megabits per second) versus GameSpy data (in players), and the least-squares fit line. The correlation coefficient for this week is 0.86, indicating a roughly linear relationship. We attribute the inexact nature of the correspondence to small changes in the size of the authentication module and sampling error.

We use the GameSpy dataset to subtract the authentication data from Steam and focus on the bandwidth requirements of a patch. Figure 16 shows a two week period of Steam activity, with a single patch occurring three days into the period. Also graphed is the authentication data component, computed from the GameSpy dataset with a ratio of players to megabits/second of 1 to 0.0291. By integrating these two signals and subtracting, we estimate the patch burden on Steam for this patch to be 129.7 terabytes, which is 30% of that week's total load including authentication.

We use this same methodology on four patches delivered during our trace, and chart the bandwidth impact of the patches over a two-week period in Figure 17. Three anomalies deserve explanation: patch *p3* is cut short of the full two week period analysis because of the release of *p5*, patch *p2* shows a rise in bandwidth after one week due to erroneous player data from GameSpy, and (according to Steam's press releases) the two weeks of patch *p7* contain numerous patches. One question to address is how long it

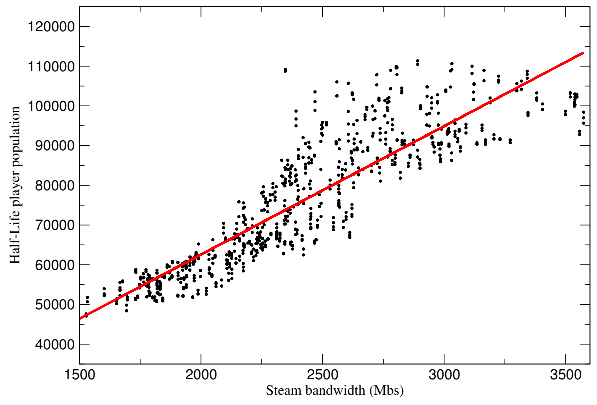


Figure 15: Half-Life player population versus Steam CDN usage

takes to deliver a patch: the cumulative distribution function (CDF) of the patch delivery data in Figure 18 shows that 80% of the load occurs in the first 72 hours for the three single-patch traces, whereas the various patches in trace *p7* are delivered throughout a two-week period.

Our observations on patch distribution bring up several issues. We believe content delivery for games is a significant burden that must be provisioned for, as it can greatly increase the hosting bandwidth requirement. At this point, however, it is unclear what the optimal strategy would be for delivery and scheduling. Our inital observations are that to avoid the stacking effect seen in Figure 18, content should be spaced for delivery such that the bulk of each patch is delivered before the next patch begins. Further, if minimizing the combined content and authentication load is a goal, then patches should be released at the lowest peak

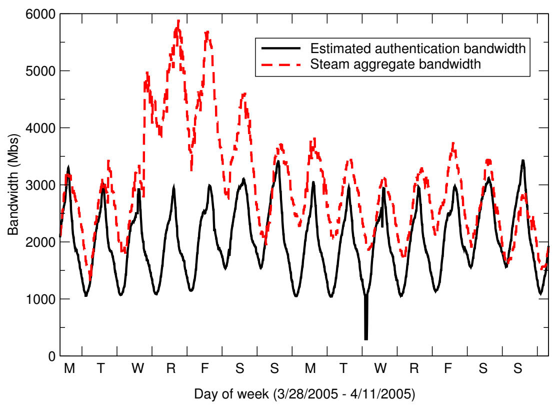


Figure 16: Steam bandwidth during a patch release

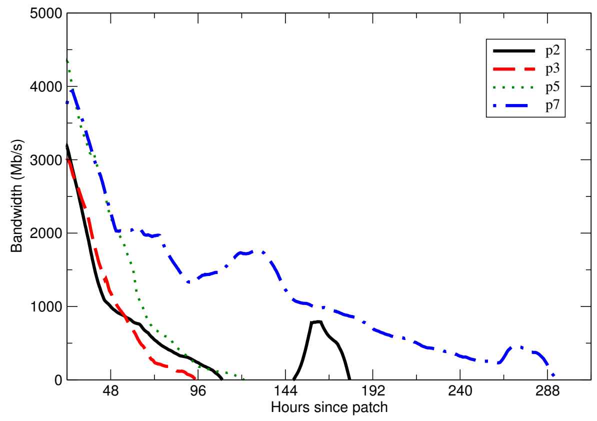


Figure 17: Excess bandwidth consumed by users downloading patches via Steam

in the weekly and daily cycle. For example, a patch released Monday evening may potentially miss the daily afternoon peak as well as the weekend peak. As part of future work, we plan on examining the proper scheduling of patches based on measured game workloads.

6 Conclusions

On-line gaming is an increasingly popular form of entertainment on the Internet. Unfortunately, effectively hosting on-line games is a difficult, expensive proposition made more onerous by the lack of workload models for games or known characteristics of gamers. Due to the unpredictable nature of the popularity of a game, combined with the high barrier to entry for hosting, a number of academic and industry projects have focused on providing a shared on-demand infrastructure to solve the hosting problem.

To understand the benefits of such infrastructure, this paper presents a comprehensive analysis of a collection of on-

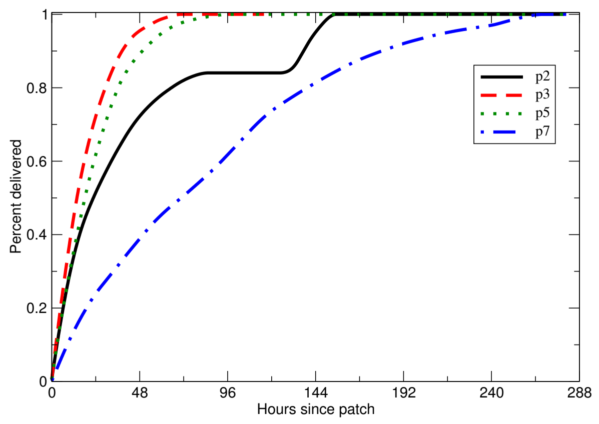


Figure 18: Cumulative distribution function of patch data.

line game players and game usage data from a number of unique sources, mostly biased towards the FPS genre. Our results show that gamers are difficult to satisfy throughout the gameplay process: they are likely to leave and never return if they can't connect, they are likely to leave within the first few minutes if they don't enjoy the server's characteristics, and they are unlikely to become loyal to a server. In addition, the popularity of this collection of games follows a power-law distribution, with a small number of games having orders of magnitude more players than the rest. This makes resource provisioning very difficult for the initial release of a game when popularity has not been established and provides a promising area where shared hosting can provide benefit. Although initial provisioning is difficult, our results also show that once established, game workloads are relatively stable from week to week, allowing game providers to more easily allocate resources to meet demand. In addition, we determine that game workloads are synchronized amongst themselves and other interactive applications and that they follow strong diurnal, geographic patterns. Such synchronization makes it difficult to obtain statistical multiplexing gain between games and other interactive applications when using shared infrastructure. Finally, we show that game software updates provide a significant burden on game hosting and must be scheduled and planned for accordingly.

References

[1] DFC Intelligence, "Online Game Market is Growing but Making Money is Difficult," `http://www.dfcint.com/news/prjune252003.html`.

[2] Sony Online Entertainment, "Sony Online Entertainment Ships Everquest: Gates of Discord,"

http://sonyonline.com/corp/press_releases/020904_GOD_ships.html, 2004.

[3] IBM Corp., "On demand business," http://www.ibm.com/ondemand.

[4] IDC, "HP utility data center: Enabling enhanced data center agility," http://www.hp.com/large/globalsolutions/ae/pdfs/udc_enabling.pdf, May 2003.

[5] Sun Microsystems, "N1 Grid – introducing just in time computing," http://wwws.sun.com/software/solutions/n1/wp-n1.pdf, 2003.

[6] GGF, "Global Grid Forum," http://www.ggf.org.

[7] IBM Corp., "Tivoli intelligent thinkdynamic orchestrator," http://www.ibm.com/software/tivoli/products/intell-orch, 2004.

[8] E. Manoel et al., *Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator*, IBM International Technical Support Organization, December 2003, http://www.redbooks.ibm.com.

[9] D. Saha, S. Sahu, and A. Shaikh, "A Service Platform for On-Line Games," in *NetGames*, Redwood City, CA 2003.

[10] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a Service Platform for Online Games," in *NetGames*, August 2004.

[11] M. Crovella and A. Bestavros, "Self-similarity in World Wide Web Traffic: Evidence and Possible Causes," in *Proceedings of ACM SIGMETRICS*, May 1996.

[12] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer Workload," in *Proceedings of ACM SOSP*, October 2003.

[13] Online Game Publisher, "Private Communication," 2004.

[14] mshmro.com, "Counter-strike Server," http://www.mshmro.com/.

[15] W. Feng, F. Chang, W. Feng, and J. Walpole, "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," in *Proc. of the Internet Measurement Workshop*, November 2002.

[16] AMDZone, "Valve Releases Hammer Port of Counter-Strike Server," http://www.amdzone.com/releaseview.cfm?ReleaseID=1050, 2003.

[17] ServerSpy, "ServerSpy.Net: World Server Ranks," http://www.serverspy.net/site/serverranks/, 2004.

[18] GameSpy Industries, "GameSpy: Gaming's Home Page," http://www.gamespy.com/, 2002.

[19] Valve, Inc., "Steam," http://www.steampowered.com/, 2005.

[20] L. Catledge and J. Pitkow, "Characterizing Browsing Strategies in the World-Wide Web," *Computer Networks and ISDN Systems*, vol. 27, no. 6, pp. 1065–1073, 1995.

[21] F. Chang and W. Feng, "Modeling Player Session Times of On-line Games," in *NetGames 2003*, May 2003.

[22] ReliaSoft Corporation, "Life Data Analysis and Reliability Engineering Theory and Principles Reference from ReliaSoft," http://www.weibull.com/lifedatawebcontents.htm, 2003.

[23] T. Henderson and S. Bhatti, "Modelling User Behavior in Networked Games," in *ACM Multimedia*, 2001, pp. 212–220.

[24] Half-Life Admin Mod Developers, "Half-Life Admin Mod Home," http://www.adminmod.org/.

[25] AMX Mod Developers, "AMX Mod Server Plugin," http://amxmod.net/.

[26] D. Papagiannaki, N. Taft, Z. Zhang, and C. Diot, "Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models," in *Proc. IEEE INFOCOM*, 2003.

[27] CNN, "SoBig.F Breaks Virus Speed Records," http://www.cnn.com/2003/TECH/internet/08/21/sobig.virus, 2003.

[28] Microsoft Corporation, "Xbox Live," http://www.xbox.com/live, 2003.

[29] Electronic Arts, Inc., "EA.com," http://www.ea.com/, 2003.

[30] Butterfly.net, Inc., "Butterfly Grid Solution for Online Games," http://www.butterfly.net/, 2003.

[31] Geobytes, Inc., "Geobytes Home Page," http://www.geobytes.com/, 2003.

A First Look at Modern Enterprise Traffic

Ruoming Pang[†], Mark Allman[‡], Mike Bennett[¶], Jason Lee[¶], Vern Paxson[‡,¶], Brian Tierney[¶]
[†]*Princeton University,* [‡]*International Computer Science Institute,*
[¶]*Lawrence Berkeley National Laboratory (LBNL)*

Abstract

While wide-area Internet traffic has been heavily studied for many years, the characteristics of traffic *inside* Internet enterprises remain almost wholly unexplored. Nearly all of the studies of enterprise traffic available in the literature are well over a decade old and focus on individual LANs rather than whole sites. In this paper we present a broad overview of internal enterprise traffic recorded at a medium-sized site. The packet traces span more than 100 hours, over which activity from a total of several thousand internal hosts appears. This wealth of data—which we are publicly releasing in anonymized form—spans a wide range of dimensions. While we cannot form general conclusions using data from a single site, and clearly this sort of data merits additional in-depth study in a number of ways, in this work we endeavor to characterize a number of the most salient aspects of the traffic. Our goal is to provide a first sense of ways in which modern enterprise traffic is similar to wide-area Internet traffic, and ways in which it is quite different.

1 Introduction

When Cáceres captured the first published measurements of a site's wide-area Internet traffic in July, 1989 [4, 5], the entire Internet consisted of about 130,000 hosts [13]. Today, the largest enterprises can have more than that many hosts just by themselves.

It is striking, therefore, to realize that more than 15 years after studies of wide-area Internet traffic began to flourish, the nature of traffic *inside* Internet enterprises remains almost wholly unexplored. The characterizations of enterprise traffic available in the literature are either vintage LAN-oriented studies [11, 9], or, more recently, focused on specific questions such as inferring the roles played by different enterprise hosts [23] or communities of interest within a site [2]. The only broadly flavored look at traffic within modern enterprises of which we are aware is the study of OSPF routing behavior in [21]. Our aim is to complement that study with a look at the make-up of traffic as seen at the packet level within a contemporary enterprise network.

One likely reason why enterprise traffic has gone unstudied for so long is that it is technically difficult to measure. Unlike a site's Internet traffic, which we can generally record by monitoring a single access link, an enterprise of significant size lacks a single choke-point for its internal traffic. For the traffic we study in this paper, we primarily recorded it by monitoring (one at a time) the enterprise's two central routers; but our measurement apparatus could only capture two of the 20+ router ports at any one time, so we could not attain any sort of comprehensive snapshot of the enterprise's activity. Rather, we piece together a partial view of the activity by recording a succession of the enterprise's subnets in turn. This piecemeal tracing methodology affects some of our assessments. For instance, if we happen to trace a portion of the network that includes a large mail server, the fraction of mail traffic will be measured as larger than if we monitored a subnet without a mail server, or if we had an ideally comprehensive view of the enterprise's traffic. Throughout the paper we endeavor to identify such biases as they are observed. While our methodology is definitely imperfect, to collect traces from a site like ours in a comprehensive fashion would require a large infusion of additional tracing resources.

Our study is limited in another fundamental way, namely that all of our data comes from a single site, and across only a few months in time. It has long been established that the wide-area Internet traffic seen at different sites varies a great deal from one site to another [6, 16] and also over time [16, 17], such that studying a single site *cannot* be representative. Put another way, for wide-area Internet traffic, the very notion of "typical" traffic is not well-defined. We would expect the same to hold for enterprise traffic (though this basic fact actually remains to be demonstrated), and therefore our single-site study can at best provide an *example* of what modern enterprise traffic looks like, rather than a general representation. For instance, while other studies have shown peer-to-peer file sharing applications to be in widespread use [20], we observe nearly none of it in our traces (which is likely a result of organizational policy).

Even given these significant limitations, however, there is much to explore in our packet traces, which span more than 100 hours and in total include activity from 8,000 internal addresses at the Lawrence Berkeley National Laboratory and 47,000 external addresses. Indeed, we found the very wide range of dimensions in which we might examine the data difficult to grapple with. Do we characterize individual applications? Transport protocol dynamics? Evidence for self-similarity? Connection locality? Variations over time? Pathological behavior? Application efficiency? Changes since previous studies? Internal versus external traffic? Etc.

Given the many questions to explore, we decided in this first look to pursue a broad overview of the characteristics of the traffic, rather than a specific question, with an aim towards informing future, more tightly scoped efforts. To this end, we settled upon the following high-level goals:

- To understand the makeup (working up the protocol stack from the network layer to the application layer) of traffic on a modern enterprise network.
- To gain a sense of the patterns of locality of enterprise traffic.
- To characterize application traffic in terms of how intranet traffic characteristics can differ from Internet traffic characteristics.
- To characterize applications that might be heavily used in an enterprise network but only rarely used outside the enterprise, and thus have been largely ignored by modeling studies to date.
- To gain an understanding of the load being imposed on modern enterprise networks.

Our general strategy in pursuing these goals is "understand the big things first." That is, for each of the dimensions listed above, we pick the most salient contributors to that dimension and delve into them enough to understand their next degree of structure, and then repeat the process, perhaps delving further if the given contributor remains dominant even when broken down into components, or perhaps turning to a different high-level contributor at this point. The process is necessarily somewhat opportunistic rather than systematic, as a systematic study of the data would consume far more effort to examine, and text to discuss, than is feasible at this point.

The general structure of the paper is as follows. We begin in § 2 with an overview of the packet traces we gathered for our study. Next, § 3 gives a broad breakdown of the main components of the traffic, while § 4 looks at the locality of traffic sources and destinations. In § 5 we examine characteristics of the applications that dominate the traffic. § 6 provides an assessment of the load carried by the monitored networks. § 7 offers final thoughts. We note that given the breadth of the topics covered in this paper, we have spread discussions of related work throughout the paper, rather than concentrating these in their own section.

	D_0	D_1	D_2	D_3	D_4
Date	10/4/04	12/15/04	12/16/04	1/6/05	1/7/05
Duration	10 min	1 hr	1 hr	1 hr	1 hr
Per Tap	1	2	1	1	1-2
# Subnets	22	22	22	18	18
# Packets	17.8M	64.7M	28.1M	21.6M	27.7M
Snaplen	1500	68	68	1500	1500
Mon. Hosts	2,531	2,102	2,088	1,561	1,558
LBNL Hosts	4,767	5,761	5,210	5,234	5,698
Remote Hosts	4,342	10,478	7,138	16,404	23,267

Table 1: Dataset characteristics.

2 Datasets

We obtained multiple packet traces from two internal network locations at the Lawrence Berkeley National Laboratory (LBNL) in the USA. The tracing machine, a 2.2 GHz PC running FreeBSD 4.10, had four NICs. Each captured a unidirectional traffic stream extracted, via network-controllable Shomiti taps, from one of the LBNL network's central routers. While the kernel did not report any packet-capture drops, our analysis found occasional instances where a TCP receiver acknowledged data not present in the trace, suggesting the reports are incomplete. It is difficult to quantify the significance of these anomalies.

We merged these streams based on timestamps synchronized across the NICs using a custom modification to the NIC driver. Therefore, with the four available NICs we could capture traffic for two LBNL subnets. A further limitation is that our vantage point enabled the monitoring of traffic to and from the subnet, but not traffic that remained within the subnet. We used an *expect* script to periodically change the monitored subnets, working through the 18–22 different subnets attached to each of the two routers.

Table 1 provides an overview of the collected packet traces. The "per tap" field indicates the number of traces taken on each monitored router port, and *Snaplen* gives the maximum number of bytes captured for each packet. For example, D_0 consists of full-packet traces from each of the 22 subnets monitored once for 10 minutes at a time, while D_1 consists of 1 hour header-only (68 bytes) traces from the 22 subnets, each monitored twice (i.e., two 1-hour traces per subnet).

	D_0	D_1	D_2	D_3	D_4
IP	99%	97%	96%	98%	96%
!IP	1%	3%	4%	2%	4%
ARP	10%	6%	5%	27%	16%
IPX	80%	77%	65%	57%	32%
Other	10%	17%	29%	16%	52%

Table 2: Fraction of packets observed using the given network layer protocol.

3 Broad Traffic Breakdown

We first take a broad look at the protocols present in our traces, examining the network, transport and application layers.

Table 2 shows the distribution of "network layer" protocols, i.e., those above the Ethernet link layer. IP dominates, constituting more than 95% of the packets in each dataset, with the two largest non-IP protocols being IPX and ARP; the distribution of non-IP traffic varies considerably across the datasets, reflecting their different subnet (and perhaps time-of-day) makeup.[1]

Before proceeding further, we need to deal with a somewhat complicated issue. The enterprise traces include *scanning traffic* from a number of sources. The most significant of these sources are legitimate, reflecting proactive vulnerability scanning conducted by the site. Including traffic from scanners in our analysis would skew the proportion of connections due to different protocols. And, in fact, scanners can engage services that otherwise remain idle, skewing not only the magnitude of the traffic ascribed to some protocol but also the number of protocols encountered.

	D_0	D_1	D_2	D_3	D_4
Bytes (GB)	13.12	31.88	13.20	8.98	11.75
TCP	66%	95%	90%	77%	82%
UDP	34%	5%	10%	23%	18%
ICMP	0%	0%	0%	0%	0%
Conns (M)	0.16	1.17	0.54	0.75	1.15
TCP	26%	19%	23%	10%	8%
UDP	68%	74%	70%	85%	87%
ICMP	6%	6%	8%	5%	5%

Table 3: Fraction of connections and bytes utilizing various transport protocols.

In addition to the known internal scanners, we identify additional scanning traffic using the following heuristic. We first identify sources contacting more than 50 distinct hosts. We then determine whether at least 45 of the distinct addresses probed were in ascending or descending order. The scanners we find with this heuristic are primarily external sources using ICMP probes, because most other external scans get blocked by scan filtering at the LBNL border. Prior to our subsequent analysis, we remove traffic from sources identified as scanners along with the 2 internal scanners. The fraction of connections removed from the traces ranges from 4–18% across the datasets. A more in-depth study of characteristics that the scanning traffic exposes is a fruitful area for future work.

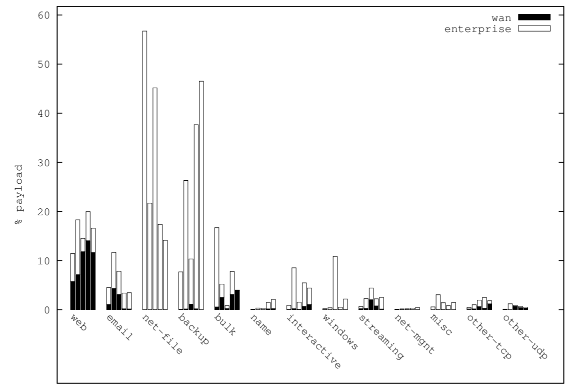
We now turn to Table 3, which breaks down the traffic by transport protocol (i.e., above the IP layer) in terms of payload bytes and packets for the three most popular transports found in our traces. The "Bytes" and "Conns" rows give the total number of payload bytes and connections for each dataset in Gbytes and millions, respectively. The ICMP traffic remains fairly consistent across all datasets, in terms of fraction of both bytes and connections. The mix of TCP and UDP traffic varies a bit more. We note that the bulk of the bytes are sent using TCP, and the bulk of the connections use UDP, for reasons explored below. Finally, we observe a number of additional transport protocols in our datasets, each of which make up only a slim portion of the traffic, including IGMP, IPSEC/ESP, PIM, GRE, and IP protocol 224 (unidentified).

Category	Protocols
backup	Dantz, Veritas, "connected-backup"
bulk	FTP, HPSS
email	SMTP, IMAP4, IMAP/S, POP3, POP/S, LDAP
interactive	SSH, telnet, rlogin, X11
name	DNS, Netbios-NS, SrvLoc
net-file	NFS, NCP
net-mgnt	DHCP, ident, NTP, SNMP, NAV-ping, SAP, NetInfo-local
streaming	RTSP, IPVideo, RealStream
web	HTTP, HTTPS
windows	CIFS/SMB, DCE/RPC, Netbios-SSN, Netbios-DGM
misc	Steltor, MetaSys, LPD, IPP, Oracle-SQL, MS-SQL

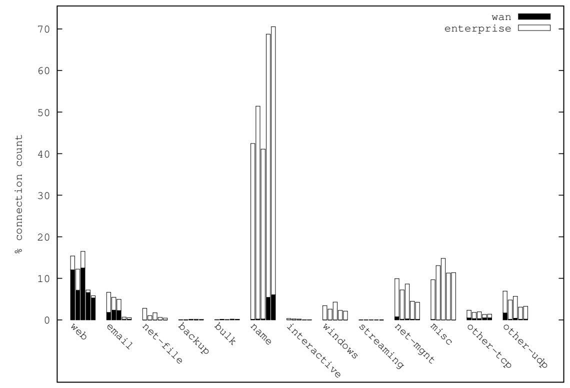
Table 4: Application categories and their constituent protocols.

Next we break down the traffic by application category. We group TCP and UDP application protocols as shown in Table 4. The table groups the applications together based on their high-level purpose. We show only those distinguished by the amount of traffic they transmit, in terms of packets, bytes or connections (we omit *many* minor additional categories and protocols). In § 5 we examine the characteristics of a number of these application protocols.

Figure 1 shows the fraction of unicast payload bytes and connections from each application category (multicast traffic is discussed below). The five bars for each category correspond to our five datasets. The total height of the bar represents the percentage of traffic due to the given category. The solid part of the bar represents the fraction of the total in which one of the endpoints of the connection resides outside of LBNL, while the hollow portion of the bar represents the fraction of the total that remains within LBNL's network. (We delve into traffic origin and locality in more depth in § 4.) We also examined the traffic breakdown in terms of packets, but since it is similar to the breakdown in terms of bytes, we do not include the plot due to space constraints. We note, however, that when measuring in terms of packets the percentage of interactive traffic is roughly a factor of two more than when assessing the



(a) Bytes



(b) Connections

Figure 1: Fraction of traffic using various application layer protocols.

traffic in terms of bytes, indicating that interactive traffic consists, not surprisingly, of small packets.

The plots show a *wider range of application usage* within the enterprise than over the WAN. In particular, we observed 3–4 times as many application categories on the internal network as we did traversing the border to the WAN. The wider range likely reflects the impact of administrative boundaries such as trust domains and firewall rules, and if so should prove to hold for enterprises in general. The figure also shows that the majority of traffic observed is local to the enterprise. This follows the familiar pattern of locality in computer and network systems which, for example, plays a part in memory, disk block, and web page caching.

In addition, Figure 1 shows the reason for the finding above that most of the connections in the traces use UDP, while most of the bytes are sent across TCP connections. Many connections are for "name" traffic across all the datasets (45–65% of the connections). However, the byte count for "name" traffic constitutes no more than 1% of the aggregate traffic. The "net-mgnt", "misc" and "other-udp" categories show similar patterns. While most of the connections are short transaction-style transfers, most of the bytes that traverse the network are from a relatively few connections. Figure 1(a) shows that the "bulk", "network-file" and "backup" categories constitute a majority of the bytes observed across datasets. In some of the datasets, "windows", "streaming" and "interactive" traffic each contribute 5–10% of the bytes observed, as well. The first two make sense because they include bulk-transfer as a component of their traffic; and in fact interactive traffic does too, in the form of SSH, which can be used not only as an interactive login facility but also for copying files and tunneling other applications.

Most of the application categories shown in Figure 1 are *unbalanced* in that the traffic is dominated by either the connection count or the byte count. The "web" and "email" traffic categories are the exception; they show non-negligible contributions to both the byte and connection counts. We will characterize these applications in detail in § 5, but here we note that this indicates that most of the traffic in these categories consists of connections with modest—not tiny or huge—lengths.

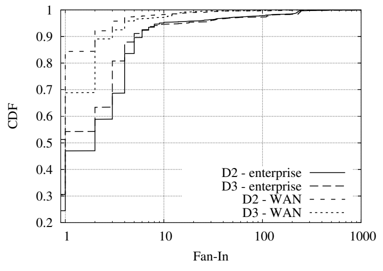
In addition, the plot highlights the differences in traffic profile across time and area of the network monitored. For instance, the number of bytes transmitted for "backup" activities varies by a factor of roughly 5 from D_0 to D_4. This could be due to differences in the monitored locations, or different tracing times. Given our data collection techniques, we cannot distill trends from the data; however this is clearly a fruitful area for future work. We note that most of the application categories that significantly contribute to the traffic mix show a range of usage across the datasets. However, the percentage of connections in the "net-mgnt" and "misc" categories are fairly consistent across the datasets. This may be because a majority of the connections come from periodic probes and announcements, and thus have a quite stable volume.

Finally, we note that multicast traffic constitutes a significant fraction of traffic in the "streaming", "name", and "net-mgnt" categories. We observe that 5–10% of all TCP/UDP payload bytes transmitted are in multicast streaming—i.e., more than the amount of traffic found in unicast streaming. Likewise, multicast traffic in "name" (SrvLoc) and "net-mgnt" (SAP) each constitutes 5–10% of all TCP/UDP connections. However, multicast traffic in the remaining application categories was found to be negligible.

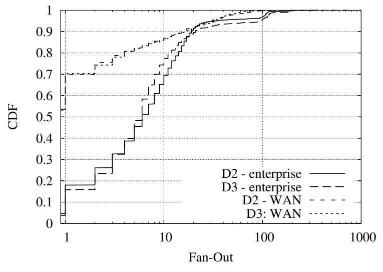
4 Origins and Locality

We next analyze the data to assess both the origins of traffic and the breadth of communications among the monitored hosts. First, we examine the origin of the flows in each dataset, finding that the traffic is clearly dominated by unicast flows whose source and destination are both within the enterprise (71–79% of flows across the five datasets). Another 2–3% of unicast flows originate within the enterprise but communicate with peers across the wide-area network, and 6–11% originate from hosts outside of the enterprise contacting peers within the enterprise. Finally, 5–10% of the flows use multicast sourced from within the enterprise and 4–7% use multicast sourced externally.

We next assess the number of hosts with which each monitored host communicates. For each monitored host H we compute two metrics: (i) *fan-in* is the number of hosts that originate conversations with H, while (ii) *fan-out* is the number of hosts to which H initiates conversations. We calculate these metrics in terms of both local traffic and wide-area traffic.



(a) Fan-in



(b) Fan-out

Figure 2: Locality in host communication.

Figure 2 shows the distribution of fan-in and fan-out for D_2 and D_3.[2] We observe that for both fan-in and fan-out, the hosts in our datasets generally have more peers within the enterprise than across the WAN, though with considerable variability. In particular, one-third to one-half of the hosts have only internal fan-in, and more than half with only internal fan-out — much more than the fraction of hosts with only external peers. This difference matches our intuition that local hosts will contact local servers (e.g., SMTP, IMAP, DNS, distributed file systems) more frequently than requesting services across the wide-area network, and is also consistent with our observation that a wider variety of applications are used only within the enterprise.

While most hosts have a modest fan-in and fan-out—over 90% of the hosts communicate with at most a couple dozen other hosts—some hosts communicate with scores to hundreds of hosts, primarily busy servers that communicate with large numbers of on- and off-site hosts (e.g., mail servers). Finally, the tail of the internal fan-out, starting around 100 peers/source, is largely due to the peer-to-peer communication pattern of SrvLoc traffic.

In keeping with the spirit of this paper, the data presented in this section provides a first look at origins and locality in the aggregate. Future work on assessing particular applications and examining locality within the enterprise is needed.

§ 5.1.1	Automated HTTP client activities constitute a significant fraction of internal HTTP traffic.
§ 5.1.2	IMAP traffic inside the enterprise has characteristics similar to wide-area email, except connections are longer-lived.
§ 5.1.3	Netbios/NS queries fail nearly 50% of the time, apparently due to popular names becoming stale.
§ 5.2.1	Windows traffic is intermingled over various ports, with Netbios/SSN (139/tcp) and SMB (445/tcp) used interchangeably for carrying CIFS traffic. DCE/RPC over "named pipes", rather than Windows File Sharing, emerges as the most active component in CIFS traffic. Among DCE/RPC services, printing and user authentication are the two most heavily used.
§ 5.2.2	Most NFS and NCP requests are reading, writing, or obtaining file attributes.
§ 5.2.3	Veritas and Dantz dominate our enterprise's backup applications. Veritas exhibits only client $\rightarrow$ server data transfers, but Dantz connections can be large in either direction.

Table 5: Example application traffic characteristics.

5 Application Characteristics

In this section we examine transport-layer and application-layer characteristics of individual application protocols. Table 5 provides a number of examples of the findings we make in this section.

We base our analysis on connection summaries generated by Bro [18]. As noted in § 2, D_1 and D_2 consist of traces that contain only the first 68 bytes of each packet. Therefore, we omit these two datasets from analyses that require in-depth examination of packet payloads to extract application-layer protocol messages.

Before turning to specific application protocols, however, we need to first discuss how we compute failure

	Request			Data		
	D0/ent	D3/ent	D4/ent	D0/ent	D3/ent	D4/ent
Total	7098	16423	15712	602MB	393MB	442MB
scan1	20%	45%	19%	0.1%	0.9%	1%
google1	23%	0.0%	1%	45%	0.0%	0.1%
google2	14%	8%	4%	51%	69%	48%
ifolder	1%	0.2%	10%	0.0%	0.0%	9%
All	58%	54%	34%	96%	70%	59%

Table 6: Fraction of internal HTTP traffic from automated clients.

rates. At first blush, counting the number of failed connections/requests seems to tell the story. However, this method can be misleading if the client is automated and endlessly retries after being rejected by a peer, as happens in the case of NCP, for example. Therefore, we instead determine the number of *distinct operations* between *distinct host-pairs* when quantifying success and failure. Such operations can span both the transport layer (e.g., a TCP connection request) and the application layer (e.g., a specific name lookup in the case of DNS). Given the short duration of our traces, we generally find a specific operation between a given pair of hosts either nearly always succeeds, or nearly always fails.

5.1 Internal/External Applications

We first investigate applications categories with traffic in both the enterprise network and in the wide-area network: web, email and name service.

5.1.1 Web

HTTP is one of the few protocols where we find more wide-area traffic than internal traffic in our datasets. Characterizing wide-area Web traffic has received much attention in the literature over the years, e.g., [14, 3]. In our first look at modern enterprise traffic, we find internal HTTP traffic to be distinct from WAN HTTP traffic in several ways: (i) we observe that automated clients—scanners, bots, and applications running on top of HTTP—have a large impact on overall HTTP traffic characteristics; (ii) we find a lower fan-out per client in enterprise web traffic than in WAN web traffic; (iii) we find a higher connection failure rate within the enterprise; and (iv) we find heavier use of HTTP's *conditional* `GET` in the internal network than in the WAN. Below we examine these findings along with several additional traffic characteristics.

Automated Clients: In internal Web transactions we find three activities not originating from traditional user-browsing: *scanners*, *Google bots*, and programs running on top of HTTP (e.g., Novell *iFolder* and Viacom *Net-Meeting*). As Table 6 shows, these activities are highly significant, accounting for 34–58% of internal HTTP requests and 59–96% of the internal data bytes carried over HTTP. Including these activities skews various HTTP characteristics. For instance, both Google bots and the scanner have a very high "fan-out"; the scanner provokes many more "404 File Not Found" HTTP replies than standard web browsing; *iFolder* clients use `POST` more frequently than regular clients; and *iFolder* replies often have a uniform size of 32,780 bytes. Therefore, while the presence of these activities is the biggest difference between internal and wide-area HTTP traffic, we exclude these from the remainder of the analysis in an attempt to understand additional differences.

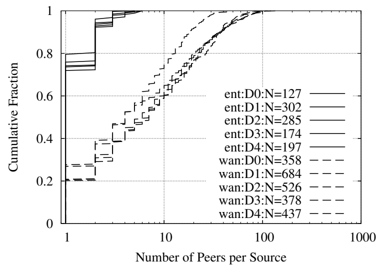


Figure 3: HTTP fan-out. The N in the key is the number of samples throughout the paper – in this case, the number of clients.

Fan-out: Figure 3 shows the distribution of fan-out from monitored clients to enterprise and WAN HTTP servers. Overall, monitored clients visit roughly an order of magnitude more external servers than internal servers. This seems to differ from the finding in § 4 that over all traffic clients tend to access more local peers than remote peers. However, we believe that the pattern shown by HTTP transactions is more likely to be the prevalent application-level pattern and that the results in § 4 are dominated by the fact that clients access a wider variety of applications. This serves to highlight the need for future work to drill down on the first, high-level analysis we present in this paper.

Connection Success Rate: Internal HTTP traffic shows success rates of 72–92% (by number of host-pairs), while the success rate of WAN HTTP traffic is 95–99%. The root cause of this difference remains a mystery. We note that the majority of unsuccessful internal connections are terminated with TCP RSTs by the servers, rather than going unanswered.

Conditional Requests: Across datasets and localities, HTTP `GET` commands account for 95–99% of both the number of requests and the number of data bytes. The `POST` command claims most of the rest. One notable difference between internal and wide area HTTP traffic is the heavier use internally of conditional `GET` commands (i.e., a `GET` request that includes one of the `If-Modified-Since` headers, per [8]). Internally we find conditional `GET` commands representing 29–53% of web requests, while externally conditional `GET` commands

	Request		Data	
	enterprise	wan	enterprise	wan
text	18% – 30%	14% – 26%	7% – 28%	13% – 27%
image	67% – 76%	44% – 68%	10% – 34%	16% – 27%
application	3% – 7%	9% – 42%	57% – 73%	33% – 60%
Other	0.0% – 2%	0.3% – 1%	0.0% – 9%	11% – 13%

Table 7: HTTP reply by content type. "Other" mainly includes *audio*, *video*, and *multipart*.

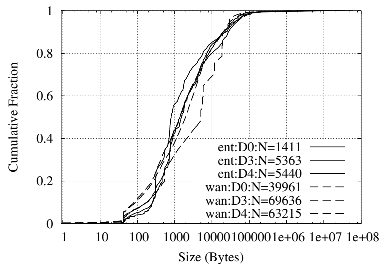


Figure 4: Size of HTTP reply, when present.

	Bytes				
	D0/all	D1/all	D2/all	D3/all	D4/all
SMTP	152MB	1658MB	393MB	20MB	59MB
SIMAP	185MB	1855MB	612MB	236MB	258MB
IMAP4	216MB	2MB	0.7MB	0.2MB	0.8MB
Other	9MB	68MB	21MB	12MB	21MB

Table 8: Email Traffic Size

account for 12–21% of the requests. The conditional requests often yield savings in terms of the number of data bytes downloaded in that conditional requests only account for 1–9% of the HTTP data bytes transfered internally and 1–7% of the data bytes transfered from external servers. We find this use of the conditional `GET` puzzling in that we would expect that attempting to save wide-area network resources (by caching and only updating content when needed) would be more important than saving local network resources. Finally, we find that over 90% of web requests are successful (meaning either the object requested is returned or that an HTTP 304 ("not modified") reply is returned in response to a conditional `GET`).

We next turn to several characteristics for which we do not find any *consistent* differences between internal and wide-area HTTP traffic.

Content Type: Table 7 provides an overview of object types for HTTP `GET` transactions that received a 200 or 206 HTTP response code (i.e., success). The *text*, *image*, and *application* content types are the three most popular, with *image* and *application* generally accounting for most of the requests and bytes, respectively. Within the *application* type, the popular subtypes include *javascript*, *octet stream*, *zip*, and *PDF*. The *other* content types are mainly *audio*, *video*, or *multipart* objects. We do not observe significant differences between internal and WAN traffic in terms of application types.

HTTP Responses: Figure 4 shows the distribution of HTTP response body sizes, excluding replies without a body. We see no significant difference between internal and WAN servers. The short vertical lines of the D_0/WAN curve reflect repeated downloading of javascripts from a particular website. We also find that about half the web sessions (i.e., downloading an entire web page) consist of one object (e.g., just an HTML page). On the other hand 10–20% of the web sessions in our dataset include 10 or more objects. We find no significant difference across datasets or server location (local or remote).

HTTP/SSL: Our data shows no significant difference in HTTPS traffic between internal and WAN servers. However, we note that in both cases there are numerous small connections between given host-pairs. For example, in D_4 we observe 795 short connections between a single pair of hosts during an hour of tracing. Examining a few at random shows that the hosts complete the SSL handshake successfully and exchange a pair of application messages, after which the client tears down the connection almost immediately. As the contents are encrypted, we cannot determine whether this reflects application level fail-and-retrial or some other phenomenon.

5.1.2 Email

Email is the second traffic category we find prevalent in both internally and over the wide-area network. As shown in Table 8, SMTP and IMAP dominate email traffic, constituting over 94% of the volume in bytes. The remainder comes from LDAP, POP3 and POP/SSL. The table shows a transition from IMAP to IMAP/S (IMAP over SSL) between D_0 and D_1, which reflects a policy change at LBNL restricting usage of unsecured IMAP.

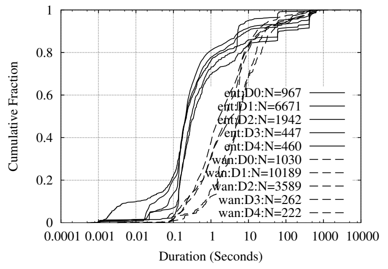
Datasets D_{0-2} include the subnets containing the main enterprise-wide SMTP and IMAP(/S) servers. This causes a difference in traffic volume between datasets D_{0-2} and D_{3-4}, and also other differences discussed below. Also, note that we conduct our analysis at the transport layer, since often the application payload is encrypted.

We note that the literature includes several studies of email traffic (e.g., [16, 10]), but none (that we are aware of) focusing on enterprise networks.

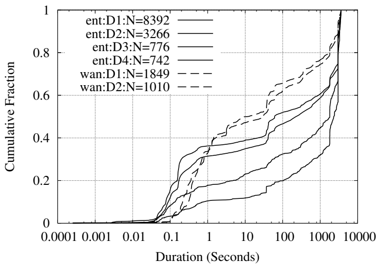
We first discuss areas where we find significant difference between enterprise and wide-area email traffic.

Connection Duration: As shown in Figure 5(a), the duration of internal and WAN SMTP connections generally differs by about an order of magnitude, with median durations around 0.2–0.4 sec and 1.5–6 sec, respectively. These results reflect the large difference in round-trip times (RTTs) experienced across the two types of network. SMTP sessions consist of both an exchange of control information and a unidirectional bulk transfer for the messages (and attachments) themselves. Both of these take time proportional to the RTT [15], explaining the longer SMTP durations.

In contrast, Figure 5(b) shows the distribution of IMAP/S connection durations across a number of our datasets. We leave off D_0 to focus on IMAP/S traffic, and D_{3-4} WAN traffic because these datasets do not include subnets with busy IMAP/S servers and hence have little wide-area IMAP/S traffic. The plot shows internal connections often last 1–2 orders of magnitude longer than wide-area connections. We do not yet have an explanation for the difference. The maximum connection duration is generally 50 minutes. While our traces are roughly 1 hour in length we find that IMAP/S clients generally poll every 10 minutes, generally providing only 5 observations within each trace. Determining the true length of IMAP/S sessions requires longer observations and is a subject for future work.



(a) SMTP

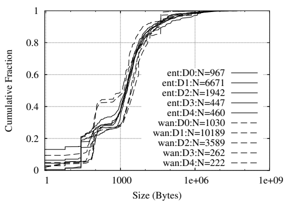


(b) IMAP/S

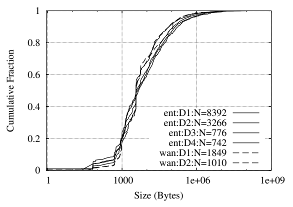
Figure 5: SMTP and IMAP/S connection durations.

We next focus on characteristics of email traffic that are similar across network type.

Connection Success Rate: Across our datasets we find that internal SMTP connections have success rates of 95–98%. SMTP connections traversing the wide-area network have success rates of 71–93% in D_{0-2} and 99-100% in D_{3-4}. Recall that D_{0-2} include heavily used SMTP servers and D_{3-4} do not, which likely explains the discrepancy. The success rate for IMAP/S connections is 99–100% across both locality and datasets.



(a) SMTP from client



(b) IMAP/S from server

Figure 6: SMTP and IMAP/S: flow size distribution

Flow Size: Internal and wide-area email traffic does not show significant differences in terms of connection sizes, as shown in Figure 6. As we would expect, the traffic volume of SMTP and IMAP/S is largely unidirectional (to SMTP servers and to IMAP/S clients), with traffic in the other direction largely being short control messages. Over 95% of the connections to SMTP servers and to IMAP/S clients remain below 1 MB, but both cases have significant upper tails.

5.1.3 Name Services

The last application category prevalent in both the internal and the wide-area traffic is domain name lookups. We observe a number of protocols providing name/directory services, including DNS, Netbios Name Service (Netbios/NS), Service Location Protocol (SrvLoc), SUN/RPC Portmapper, and DCE/RPC endpoint mapper. We also note that wide-area DNS has been studied by a number of researchers previously (e.g., [12]), however, our study of name lookups includes both enterprise traffic and non-DNS name services.

In this section we focus on DNS and Netbios/NS traffic, due to their predominant use. DNS appears in both wide-area and internal traffic. We find no large differences between the two types of DNS traffic except in response latency.

For both services a handful of servers account for most of the traffic, therefore the vantage point of the monitor can significantly affect the traffic we find in a trace. In particular, D_{0-2} do not contain subnets with a main DNS server, and thus relatively few WAN DNS connections. Therefore, in the following discussion we only use D_{3-4} for WAN DNS traffic. Similarly, more than 95% of Netbios/NS requests go to one of the two main servers. D_{0-2} captures all traffic to/from one of these and D_{3-4} captures all traffic to both. Finally, we do not consider D_{1-2} in our analysis due to the lack of application payloads in those datasets (which renders our payload analysis inoperable).

Given those considerations, we now explore several characteristics of name service traffic.

Latency: We observe median latencies are roughly 0.4 msec and 20 msec for internal and external DNS queries, respectively. This expected result is directly attributable to the round-trip delay to on- and off-site DNS servers. Netbios/NS, on the other hand, is primarily used within the enterprise, with inbound requests blocked by the enterprise at the border.

Clients: A majority of DNS requests come from a few clients, led by two main SMTP servers that perform lookups in response to incoming mail. In contrast, we find Netbios/NS requests more evenly distributed among clients, with the top ten clients generating less than 40% of all requests across datasets.

Request Type: DNS request types are quite similar both across datasets and location of the peer (internal or remote). A majority of the requests (50–66%) are for `A` records, while 17–25% are for `AAAA` (IPv6) records, which seems surprisingly high, though we have confirmed a similar ratio in the wide-area traffic at another site. Digging deeper reveals that a number of hosts are configured to request both `A` and `AAAA` in parallel. In addition, we find 10–18% of the requests are for `PTR` records and 4–7% are for `MX` records.

Netbios/NS traffic is also quite similar across the datasets. 81–85% of requests consist of name queries, with the other prevalent action being to "refresh" a registered name (12–15% of the requests). We observe a number of additional transaction types in small numbers, including commands to register names, release names, and check status.

Netbios/NS Name Type: Netbios/NS includes a "type" indication in queries. We find that across our datasets 63–71% of the queries are for workstations and servers, while 22–32% of the requests are for domain/browser information.

Return Code: We find DNS has similar success (`NOERROR`) rates (77–86%) and failure (`NXDOMAIN`) rates (11–21%) across datasets and across internal and wide-area traffic. We observe failures with Netbios/NS 2–3 times more often: 36–50% of distinct Netbios/NS queries yield an `NXDOMAIN` reply. These failures are broadly distributed—they are not due to any single client, server, or query string. We speculate that the difference between the two protocols may be attributable to DNS representing an administratively controlled name space, while Netbios/NS uses a more distributed and loosely controlled mechanism for registering names, resulting in Netbios/NS names going "out-of-date" due to timeouts or revocations.

5.2 Enterprise-Only Applications

The previous subsection deals with application categories found in both internal and wide-area communication. In this section, we turn to analyzing the high-level and salient features of applications used only within the enterprise. Given the degree to which such protocols have not seen much exploration in the literature before, we aim for a broad rather than deep examination. A great deal remains for future work to develop the characterizations in more depth.

5.2.1 Windows Services

We first consider those services used by Windows hosts for a wide range of tasks, such as Windows file sharing, authentication, printing, and messaging. In particular, we examine Netbios Session Services (SSN), the Common Internet File System (SMB/CIFS), and DCE/RPC. We do not tackle the Netbios Datagram Service since it appears to be largely used *within* subnets (e.g., for "Network Neighborhoods"), and does not appear much in our datasets; and we cover Netbios/NS in § 5.1.3.

One of the main challenges in analyzing Windows traffic is that each communication scheme can be used in a variety of ways. For instance, TCP port numbers reveal little about the actual application: services can be accessed via multiple channels, and a single port can be shared by a variety of services. Hosts appear to interchangeably use CIFS via its well-known TCP port of 445 and via layering on top of Netbios/SSN (TCP port 139). Similarly, we note that DCE/RPC clients have two ways to find services: (i) using "named pipes" on top of CIFS (which may or may not be layered on top of Netbios/SSN) and (ii) on top of standard TCP and UDP connections without using CIFS, in which case clients consult the Endpoint Mapper to discover the port of a particular DCE/RPC service. Thus, in order to understand the Windows traffic we had to develop rich Bro protocol analyzers, and also merge activities from different transport layer channels. With this in place, we could then analyze various facets of the activities according to application functionality, as follows.

Connection Success Rate: As shown in Table 9, we observe a variety of connection success rates for different kinds of traffic: 82–92% for Netbios/SSN connections, 99–100% for Endpoint Mapper traffic, and a strikingly low 46–68% for CIFS traffic. For both Netbios/SSN and CIFS traffic we find the failures are not caused by a few erratic hosts, but rather are spread across hundreds of clients and dozens of servers. Further investigation reveals most of CIFS connection failures are caused by a number of clients connecting to servers on both the Netbios/SSN (139/tcp) and CIFS (445/tcp) port *in parallel*—since the two ports can be used interchangeably. The apparent intention is to use whichever port works while not incurring the cost of trying each in turn. We also find a number of the servers are configured to listen only on the Netbios/SSN port, so they reject connections to the CIFS port.

	Host Pairs		
	Netbios/SSN	CIFS	Endpoint Mapper
Total	595 – 1464	373 – 732	119 – 497
Successful	82% – 92%	46% – 68%	99% – 100%
Rejected	0.2% – 0.8%	26% – 37%	0.0% – 0.0%
Unanswered	8% – 19%	5% – 19%	0.2% – 0.8%

Table 9: Windows traffic connection success rate (by number of host-pairs, for internal traffic only)

Netbios/SSN Success Rate: After a connection is established, a Netbios/SSN session goes through a handshake before carrying traffic. The success rate of the handshake (counting the number of distinct host-pairs) is 89–99% across our datasets. Again, the failures are not due to any single client or server, but are spread across a number of hosts. The reason for these failures merits future investigation.

CIFS Commands: Table 10 shows the prevalence of various types of commands used in CIFS channels across our datasets, in terms of both the number of commands and volume of data transferred. The first category, "SMB Basic", includes common commands used for session initialization and termination, and accounts for 24–52% of the messages across the datasets, but only 3–15% of the data bytes. The remaining categories indicate the tasks CIFS connections are used to perform. Interestingly, we find DCE/RPC pipes, rather than Windows File Sharing, make up the largest portion of messages (33–48%) and data bytes (32–77%) across datasets. Windows File Sharing constitutes 11–27% of messages and 8% to 43% of bytes. Finally, "LANMAN", a non-RPC named pipe for management tasks in "network neighborhood" systems, accounts for just 1–3% of the requests, but 3–15% of the bytes.

DCE/RPC Functions: Since DCE/RPC constitutes an important part of Windows traffic, we further analyze these calls over both CIFS pipes and stand-alone TCP/UDP connections. While we include all DCE/RPC activities traversing CIFS pipes, our analysis for DCE/RPC over stand-alone TCP/UDP connections may be incomplete for two reasons. First, we identify DCE/RPC activities on ephemeral ports by analyzing Endpoint Mapper traffic. Therefore, we will miss traffic if the mapping takes place before our trace collection begins, or if there is an alternate method to discover the server's ports (though we are not aware of any other such method). Second, our analysis tool currently cannot parse DCE/RPC messages sent over UDP. While this may cause our analysis to miss services that only use UDP, DCE/RPC traffic using UDP accounts for only a small fraction of all DCE/RPC traffic.

Table 11 shows the breakdown of DCE/RPC functions. Across all datasets, the *Spoolss* printing functions—and `WritePrinter` in particular—dominate the overall traffic in D_{3-4}, with 63–91% of the requests and 94–99% of the data bytes. In D_0, *Spoolss* traffic remains significant, but not as dominant as user authentication functions (*NetLogon* and *LsaRPC*), which account for 68% of the requests and 52% of the bytes. These figures illustrate the variations present within the enterprise, as well as highlighting the need for multiple vantage points when monitoring. (For instance, in D_0 we monitor a major authentication server, while D_{3-4} includes a major print server.)

5.2.2 Network File Services

NFS and NCP[3] comprise the two main network file system protocols seen within the enterprise and this traffic is nearly always confined to the enterprise.[4] We note that several trace-based studies of network file system characteristics have appeared in the filesystem literature (e.g., see [7] and enclosed references). We now investigate several aspects of network file system traffic.

Aggregate Sizes: Table 12 shows the number of NFS and NCP connections and the amount of data transferred for each dataset. In terms of connections, we find NFS more prevalent than NCP, except in D_0. In all datasets, we find NFS transfers more data bytes per connection than NCP. As in previous sections, we see the impact of the measurement location in that the relative amount of NCP traffic is much higher in D_{0-2} than in D_{3-4}. Finally, we find "heavy hitters" in NFS/NCP traffic: the three most active NFS host-pairs account for 89–94% of the data transfered, and for the top three NCP host-pairs, 35–62%.

Keep-Alives: We find that NCP appears to use TCP keep-alives to maintain long-lived connections and detect runaway clients. Particularly striking is that 40–80% of the NCP connections across our datasets consist *only* of periodic retransmissions of 1 data byte and therefore do not include any real activity.

UDP vs. TCP We had expected that NFS-over-TCP would heavily dominate modern use of NFS, but find this is not the case. Across the datasets, UDP comprises 66%/16%/31%/94%/7% of the payload bytes, an enormous range. Overall, 90% of the NFS host-pairs use UDP, while only 21% use TCP (some use both).

Request Success Rate: If an NCP connection attempt succeeds (88–98% of the time), about 95% of the subsequent requests also succeed, with the failures dominated by "File/Dir Info" requests. NFS requests succeed 84% to 95% of the time, with most of the unsuccessful requests being "lookup" requests for non-existing files or directories.

Requests per Host-Pair: Since NFS and NCP both use a message size of about 8 KB, multiple requests are needed for large data transfers. Figure 7 shows the number of requests per client-server pair. We see a large range, from a handful of requests to hundreds of thousands of requests between a host-pair. A related observation is that the interval between requests issued by a client is generally 10 msec

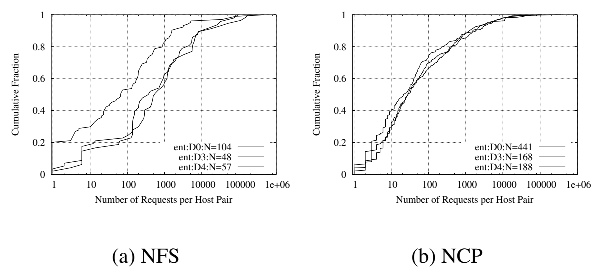
	Request			Data		
	D0/ent	D3/ent	D4/ent	D0/ent	D3/ent	D4/ent
Total	49120	45954	123607	18MB	32MB	198MB
SMB Basic	36%	52%	24%	15%	12%	3%
RPC Pipes	48%	33%	46%	32%	64%	77%
Windows File Sharing	13%	11%	27%	43%	8%	17%
LANMAN	1%	3%	1%	10%	15%	3%
Other	2%	0.6%	1.0%	0.2%	0.3%	0.8%

Table 10: CIFS command breakdown. "SMB basic" includes the common commands shared by all kinds of higher level applications: protocol negotiation, session setup/tear-down, tree connect/disconnect, and file/pipe open.

	Request			Data		
	D0/ent	D3/ent	D4/ent	D0/ent	D3/ent	D4/ent
Total	14191	13620	56912	4MB	19MB	146MB
NetLogon	42%	5%	0.5%	45%	0.9%	0.1%
LsaRPC	26%	5%	0.6%	7%	0.3%	0.0%
Spoolss/WritePrinter	0.0%	29%	81%	0.0%	80%	96%
Spoolss/other	24%	34%	10%	42%	14%	3%
Other	8%	27%	8%	6%	4%	0.6%

Table 11: DCE/RPC function breakdown.

or less.

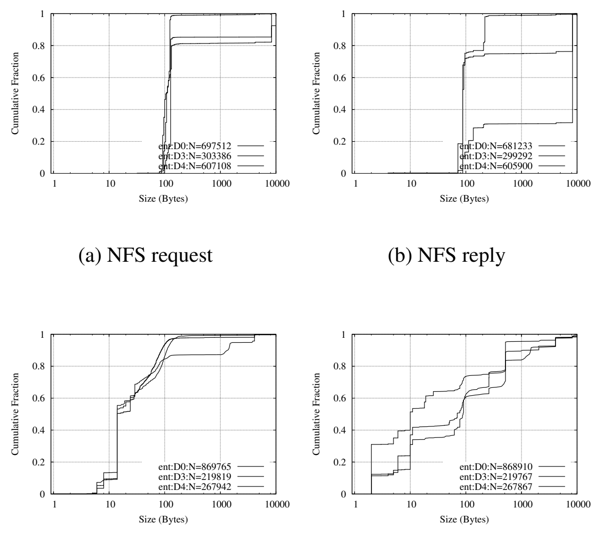


(a) NFS (b) NCP

Figure 7: NFS/NCP: number of requests per client-server pair, for those with at least one request seen.

Breakdown by Request Type: Table 13 and 14 show that in both NFS and NCP, file read/write requests account for the vast majority of the data bytes transmitted, 88–99% and 92–98% respectively. In terms of the number of requests, obtaining file attributes joins read and write as a dominant function. NCP file searching also accounts for 7–16% of the requests (but only 1–4% of the bytes). Note that NCP provides services in addition to remote file access, e.g., directory service (NDS), but, as shown in the table, in our datasets NCP is predominantly used for file sharing.

Request/Reply Data Size Distribution: As shown in Figure 8(a,b), NFS requests and replies have clear dual-mode distributions, with one mode around 100 bytes and the other 8 KB. The latter corresponds to write requests and read replies, and the former to everything else. NCP requests exhibit a mode at 14 bytes, corresponding to read requests, and each vertical rise in the NCP reply size figure corresponds to particular types of commands: 2-byte replies for completion codes only (e.g. replying to "WriteFile" or reporting error), 10 bytes for "GetFileCurrentSize", and 260 bytes for (a fraction of) "ReadFile" requests.



(a) NFS request (b) NFS reply

(c) NCP request (d) NCP reply

Figure 8: NFS/NCP: request/reply data size distribution (NFS/NCP message headers are not included)

5.2.3 Backup

Backup sessions are a rarity in our traces, with just a small number of hosts and connections responsible for a huge data volume. Clearly, this is an area where we need longer traces. That said, we offer brief characterizations here to convey a sense of its nature.

	Connections					Bytes				
	D0/all	D1/all	D2/all	D3/all	D4/all	D0/all	D1/all	D2/all	D3/all	D4/all
NFS	1067	5260	4144	3038	3347	6318MB	4094MB	3586MB	1030MB	1151MB
NCP	2590	4436	2892	628	802	777MB	2574MB	2353MB	352MB	233MB

Table 12: NFS/NCP Size

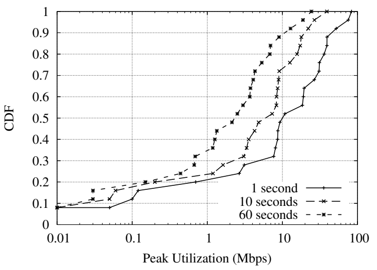
	Request			Data		
	D0/ent	D3/ent	D4/ent	D0/ent	D3/ent	D4/ent
Total	697512	303386	607108	5843MB	676MB	1064MB
Read	70%	25%	1%	64%	92%	6%
Write	15%	1%	19%	35%	2%	83%
GetAttr	9%	53%	50%	0.2%	4%	5%
LookUp	4%	16%	23%	0.1%	2%	4%
Access	0.5%	4%	5%	0.0%	0.4%	0.6%
Other	2%	0.9%	2%	0.1%	0.2%	1%

Table 13: NFS requests breakdown.

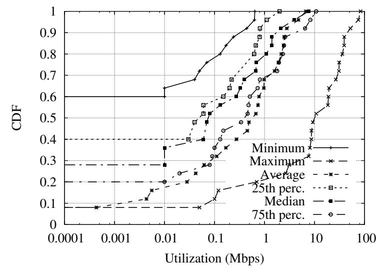
	Connections	Bytes
VERITAS-BACKUP-CTRL	1271	0.1MB
VERITAS-BACKUP-DATA	352	6781MB
DANTZ	1013	10967MB
CONNECTED-BACKUP	105	214MB

Table 15: Backup Applications

We find three types of backup traffic, per Table 15: two internal traffic giants, Dantz and Veritas, and a much smaller, "Connected" service that backs up data to an external site. Veritas backup uses separate control and data connections, with the data connections in the traces all reflecting one-way, client-to-server traffic. Dantz, on the other hand, appears to transmit control data within the same connection, and its connections display a degree of bi-directionality. Furthermore, the server-to-client flow sizes can exceed 100 MB. This bi-directionality does not appear to reflect backup vs. restore, because it exists not only *between* connections, but also *within* individual connections—sometimes with tens of MB in both directions. Perhaps this reflects an exchange of fingerprints used for compression or incremental backups or an exchange of validation information after the backup is finished. Alternatively, this may indicate that the protocol itself may have a peer-to-peer structure rather than a strict server/client delineation. Clearly this requires further investigation with longer trace files.



(a) Peak Utilization



(b) Utilization

Figure 9: Utilization distributions for D_4.

6 Network Load

A final aspect of enterprise traffic in our preliminary investigation is to assess the load observed within the enterprise. One might naturally assume that campus networks are underutilized, and some researchers aim to develop mechanisms that leverage this assumption [19]. We assess this assumption using our data.

Due to limited space, we discuss only D_4, although the other datasets provide essentially the same insights about utilization. Figure 9(a) shows the distribution of the *peak* bandwidth usage over 3 different timescales for each trace in the D_4 dataset. As expected, the plot shows the networks to be less than fully utilized at each timescale. The 1 second interval does show network saturation (100 Mbps) in some cases. However, as the measurement time interval increases the peak utilization drops, indicating that saturation is short-lived.

Figure 9(b) shows the distributions of several metrics calculated over 1 second intervals. The "maximum" line on this plot is the same as the "1 second" line on the previous

	Request			Data		
	D0/ent	D3/ent	D4/ent	D0/ent	D3/ent	D4/ent
Total	869765	219819	267942	712MB	345MB	222MB
Read	42%	44%	41%	82%	70%	82%
Write	1%	21%	2%	10%	28%	11%
FileDirInfo	27%	16%	26%	5%	0.9%	3%
File Open/Close	9%	2%	7%	0.9%	0.1%	0.5%
File Size	9%	7%	5%	0.2%	0.1%	0.1%
File Search	9%	7%	16%	1%	0.6%	4%
Directory Service	2%	0.7%	1%	0.7%	0.1%	0.4%
Other	3%	3%	2%	0.2%	0.1%	0.1%

Table 14: NCP requests breakdown.

plot. The second plot concretely shows that typical (over time) network usage is 1–2 orders of magnitude less than the peak utilization and 2–3 orders less than the capacity of the network (100 Mbps).

We can think of packet loss as a second dimension for assessing network load. We can form estimates of packet loss rates using TCP retransmission rates. These two might not fully agree, due to (i) TCP possibly retransmitting unnecessarily, and (ii) TCP adapting its rate in the presence of loss, while non-TCP traffic will not. But the former should be rare in LAN environments (little opportunity for retransmission timers to go off early), and the latter arguably at most limits our analysis to applying to the TCP traffic, which dominates the load (cf. Table 3).

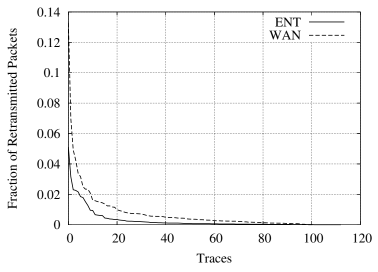


Figure 10: TCP Retransmission Rate Across Traces (for traces with at least 1000 packets in the category)

We found a number of spurious 1 byte retransmissions due to TCP keep-alives by NCP and SSH connections. We exclude these from further analysis because they do not indicate load imposed on network elements. Figure 10 shows the remaining retransmission rate for each trace in all our datasets, for both internal and remote traffic. In the vast majority of the traces, the retransmission rate remains less than 1% for both. In addition, the retransmission rate for internal traffic is less than that of traffic involving a remote peer, which matches our expectations since wide-area traffic traverses more shared, diverse, and constrained networks than does internal traffic. (While not shown in the Figure, we did not find any correlation between internal and wide-area retransmission rates.)

We do, however, find that the internal retransmission rate sometimes eclipses 2%—peaking at roughly 5% in one of the traces. Our further investigation of this last trace found the retransmissions dominated by a single Veritas backup connection, which transmitted 1.5 M packets and 2 GB of data from a client to a server over one hour. The retransmissions happen almost evenly over time, and over one-second intervals the rate peaks at 5 Mbps with a 95^{th} percentile around 1 Mbps. Thus, the losses appear due to either significant congestion in the enterprise network downstream from our measurement point, or a network element with flaky NIC (reported in [22] as not a rare event).

We can summarize these findings as: packet loss within an enterprise appears to occur significantly less than across the wide-area network, as expected; but exceeds 1% a non-negligible proportion of the time.

7 Summary

Enterprise networks have been all but ignored in the modern measurement literature. Our major contribution in this paper is to provide a broad, high-level view of numerous aspects of enterprise network traffic. Our investigation runs the gamut from re-examining topics previously studied for wide-area traffic (e.g., web traffic), to investigating new types of traffic not assessed in the literature to our knowledge (e.g., Windows protocols), to testing assumptions about enterprise traffic dynamics (e.g., that such networks are mostly underutilized).

Clearly, our investigation is only an initial step in this space. An additional hope for our work is to inspire the community to undertake more in-depth studies of the raft of topics concerning enterprise traffic that we could only examine briefly (or not at all) in this paper. Towards this end, we are releasing anonymized versions of our traces to the community [1].

Acknowledgments

We thank Sally Floyd for interesting discussions that led to § 6, Craig Leres for help with several tracing issues and Martin Arlitt and the anonymous reviewers for useful comments on a draft of this paper. Also, we gratefully acknowledge funding support for this work from NSF grants 0335241, 0205519, and 0335214, and DHS grant HSHQPA4X03322.

References

[1] LBNL Enterprise Trace Repository, 2005. http://www.icir.org/enterprise-tracing/.

[2] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck, and K. van der Merwe. Analysis of Communities Of Interest in Data Networks. In *Proceedings of Passive and Active Measurement Workshop*, Mar. 2005.

[3] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS*, pages 151–160, July 1998.

[4] R. Cáceres. Measurements of wide area internet traffic. Technical report, 1989.

[5] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel. Characteristics of Wide-Area TCP/IP Conversations. In *ACM SIGCOMM*, 1991.

[6] P. Danzig, S. Jamin, R. Cáceres, D. Mitzel, and D. Estrin. An Empirical Workload Model for Driving Wide-area TCP/IP Network Simulations. *Internetworking: Research and Experience*, 3(1):1–26, 1992.

[7] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS Tracing of Email and Research Workloads. In *USENIX Conference on File and Storage Technologies*, 2003.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, jun. RFC 2616.

[9] H. J. Fowler and W. E. Leland. Local area network traffic characteristics, with implications for broadband network congestion management. *IEEE Journal on Selected Areas in Communications*, SAC-9:1139–49, 1991.

[10] L. Gomes, C. Cazita, J. Almeida, V. Almeida, and W. M. Jr. Characterizing a SPAM Traffic. In *Internet Measurement Conference*, Oct. 2004.

[11] R. Gusella. A measurement study of diskless workstation traffic on an Ethernet. *IEEE Transactions on Communications*, 38(9):1557–1568, Sept. 1990.

[12] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and the Effectiveness of Caching. In *ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[13] M. Lottor. Internet Growth (1981-1991), Jan. 1992. RFC 1296.

[14] B. Mah. An Empirical Model of HTTP Network Traffic. In *Proceedings of INFOCOM 97*, Apr. 1997.

[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, Sept. 1998.

[16] V. Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, Aug. 1994.

[17] V. Paxson. Growth Trends in Wide-Area TCP Connections. *IEEE Network*, 8(4):8–17, July/August 1994.

[18] V. Paxson. Bro: A system for detecting network intruders in real time. *Computer Networks*, December 1999.

[19] P. Sarolahti, M. Allman, and S. Floyd. Evaluating Quick-Start for TCP. May 2005. Under submission.

[20] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. In *Internet Measurement Workshop*, pages 137–150, Nov. 2002.

[21] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of OSPF behavior in a large enterprise network. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 217–230, New York, NY, USA, 2002. ACM Press.

[22] J. Stone and C. Partridge. When The CRC and TCP Checksum Disagree. In *ACM SIGCOMM*, Sept. 2000.

[23] G. Tan, M. Poletto, J. Guttag, and F. Kaashoek. Role Classification of Hosts within Enterprise Networks Based on Connection Patterns. In *Proceedings of USENIX Annual Technical Conference*, June 2003.

Notes

[1] Hour-long traces we made of $\approx$ 100 individual hosts (not otherwise analyzed here) have a makeup of 35–67% *non*-IPv4 packets, dominated by *broadcast* IPX and ARP. This traffic is mainly confined to the host's subnet and hence not seen in our inter-subnet traces. However, the traces are too low in volume for meaningful generalization.

[2]Note, the figures in this paper are small due to space considerations. However, since we are focusing on high-level notions in this paper we ask the reader to focus on the general shape and large differences illustrated rather than the small changes and minor details (which are difficult to discern given the size of the plots).

[3] NCP is the *Netware Control Protocol*, a veritable kitchen-sink protocol supporting hundreds of message types, but primarily used within the enterprise for file-sharing and print service.

[4]We found three NCP connections with remote hosts across all our datasets!

Client Behavior and Feed Characteristics of RSS, a Publish-Subscribe System for Web Micronews

Hongzhou Liu Venugopalan Ramasubramanian Emin Gün Sirer
Department of Computer Science, Cornell University, Ithaca, NY 14853
{liuhz, ramasv, egs}@cs.cornell.edu

Abstract

While publish-subscribe systems have attracted much research interest since the last decade, few established benchmarks have emerged, and there has been little characterization of how publish-subscribe systems are used in practice. This paper examines RSS, a newly emerging, widely used publish-subscribe system for Web micronews. Based on a trace study spanning 45 days at a medium-size academic department and periodic polling of approximately 100,000 RSS feeds, we extract characteristics of RSS content and usage. We find that RSS workload resembles the Web in content size and popularity; feeds are typically small (less than 10KB), albeit with a heavy tail, and feed popularity follows a power law distribution. The update rate of RSS feeds is widely distributed; 55% of RSS feeds are updated hourly, while 25% show no updates for several days. And, only small portions of RSS content typically change during an update; 64% of updates involve less than three lines of the RSS content. Overall, this paper presents an analysis of RSS, the first widely deployed publish-subscribe system, and provides insights for the design of next generation publish-subscribe systems.

1 Introduction

Publish-subscribe or *pub-sub* systems [1, 5, 6, 8, 9, 12, 13, 15] are gaining wide acceptance with applications spanning information delivery, sensor monitoring, auction systems, and air traffic control. Previous research in this area has focused on aspects, such as system architecture, event notification, and content filtering algorithms, but has left a fundamental issue untackled, namely what does the workload of a pub-sub system look like and how do clients use pub-sub systems in practice?

This paper answers these questions by examining RSS, the first widely deployed pub-sub system, which is used for disseminating Web micronews. The architecture of RSS is quite simple: clients subscribe to a feed that they are interested in and poll the feed periodically to receive updates. RSS content is encoded in XML and displayed by a feed-reader or an RSS-integrated Web browser on the client host. Most news media support RSS feeds, and information such as announcements on Web sites and updates to weblogs, is typically disseminated through RSS. Integration into Web browsers has recently made RSS accessible to Internet users and greatly increased the popularity of RSS.

In this paper, we study the feed characteristics and client behavior in the RSS system using data collected through a combination of passive logging and active polling. First, we recorded a 45-day trace from the Department of Computer Science at Cornell University. The trace consists of 158 different RSS users, who subscribe to 667 feeds in total. We use this trace to examine the characteristics of RSS workload, such as the popularity of RSS feeds, and user behavior, including polling rate and subscription patterns. Second, we collected snapshots of RSS content by actively polling every hour 99,714 feeds listed in the feed directory *syndic8.com*. We use the feed snapshots to distill content properties, such as feed size and format, and to analyze updates in terms of update rate and amount of change.

Our study provides several insights, some expected and a few surprising, into the characteristics of the RSS system. First, we find that the RSS workload bears resemblance to the Web workload. The popularity of RSS feeds is heavy-tailed and follows a Zipf distribution similar to Web objects [3]. The typical sizes of RSS feeds are comparable to Web objects, although extremely large RSS documents are rare; most RSS feeds range from 1KB to 10KB, with a median of 5.8 KB, compared to a median of about 4 KB for the Web [7].

Second, RSS content changes more often than Web objects, albeit with a wide distribution of update rates. Our periodic snapshots show that 55% of RSS feeds update within an hour, while 25% do not change at all during 84 hours of polling. Even though RSS content may change rapidly, the update, surprisingly, affects only a tiny fraction of the content; 64% of updates involve no more than three lines of the XML. RSS clients can save a tremendous 93% of bandwidth by fetching the "delta"s instead of the entire feed during polling.

Finally, our study reveals interesting behaviors of RSS users. We find that over a third of the clients fetch feeds manually and do not use automated RSS tools that poll and check for updates periodically. Among the rest, over a half of clients poll feeds hourly, which is the default setting of

Trace length	45 days
Number of clients	158
Number of feeds	667
Number of requests	61935

Table 1: **Summary of User Traces: Clients are identified by a secure cryptographic hash of their IPs.**

most RSS readers. These behaviors indicate that enabling RSS content servers to provide feed specific polling rates to RSS readers is a more efficient way to customize RSS polling than expecting clients to configure their readers.

The rest of this paper is organized as follows: The next section provides some background on pub-sub systems and RSS. Section 3 describes our methods for studying the RSS system, and Section 4 details the results of our study. Finally, we discuss the implications of our study and conclude in Section 5.

2 Background and Related Work

Publish-subscribe systems have raised considerable interest in the research community over the years. In this section, we provide some background on pub-sub systems, a brief overview of RSS, and then summarize related research in this area.

Publish-Subscribe Systems

Publish-subscribe is a distributed computing paradigm that consists of three principal components: subscribers, publishers, and an infrastructure for event delivery. Subscribers express their interest in an event or a pattern of events. Publishers generate events. The infrastructure is responsible for matching events with the interests and sending them to the subscribers that registered the interests. Based on the way the subscribers specify their interest, pub-sub systems can be classified into two categories: topic-based and content-based. In topic-based pub-sub systems, subscribers specify their interest by subscribing to a *feed*, also known as *topic*, *channel*, *subject*, or *group*. Each event produced by the publisher is labeled with a topic and sent to all the subscribers that subscribed to this topic. In other words, publishers and subscribers are connected together by a predefined topic. The major disadvantage of topic-based systems is their expressiveness: all the topics are defined by the publisher and subscribers cannot further distinguish between events on a given topic. Content-based systems fix this problem. In such systems, subscribers specify their interest through event filters, which are functions of event contents. Published events are matched against the filters and sent to the subscribers if they match the specified filters.

2.1 RSS

RSS is a Web content syndication system [14] concerned with the propagation of XML documents containing short descriptions of Web news. The XML documents are accessed via HTTP through URLs, and the URL for a particular XML document identifies the *RSS feed*. Client applications called *RSS readers* check the contents of RSS feeds periodically and automatically on the user's behalf and display the returned results. Most feed readers poll RSS feeds once per hour by default. Newer versions of RSS support features such as *TTL*, *SkipDay*, and *SkipHour*, which help RSS readers to decide when and how often to poll the feeds. Nevertheless, most RSS providers post a rate limit to prevent aggressive readers from overloading their servers.

The RSS system is a simple topic-based pub-sub system. Publishers publish their news by putting it into an RSS feed and providing the URL for the feed on their website. RSS users subscribe to a RSS feed by specifying its URL to their RSS readers. Thereafter, the RSS readers will poll the feed periodically and display the updates to the users.

Polling period	84 hours
Number of feeds	99714
Number of snapshots	3682043
Bytes received	57GB

Table 2: **Summary of Active Polling: Feeds were polled in hourly intervals.**

2.2 Related work

Previous work on pub-sub systems has focused on the design and implementation of efficient event delivery mechanisms. Isis [8], Linda spaces [5], T-space [9], SIENA [6], Gryphon [12], TIBCO [13], Astrolabe [15], and Herald [4] are examples of pub-sub systems proposed in the past. The Joint Battlespace Infosphere project [1] is a similar effort by the Air Force to provide a pub-sub based event notification and data repository system for very large scale deployment. FeedTree [11] and CorONA [10] are recently proposed systems designed to alleviate the load on RSS feed providers by cooperative polling using a distributed hash table for coordination. While a vast amount research material on pub-sub systems are available, this is the first measurement study of a widely-deployed pub-sub system.

3 Measurement Methodology

We investigate the characteristics of the RSS system from data collected through two techniques: passively logging a 45-day user activity at the Department of Computer Science, Cornell University and actively polling nearly 100,000 feeds every hour for 84 hours. The rest of this section describes how we gathered the RSS data.

Passive Logging

We built a software tool for tracing RSS traffic and installed it at the network border of our department. Our department is a medium-size academic organization with about 600 graduate students, faculty, and staff. The network is topologically separated from transient users, such as undergraduates in computer labs, who do not have dedicated computers for long-running programs. We traced user activity over a 45 day period, spanning from 22 March to 3

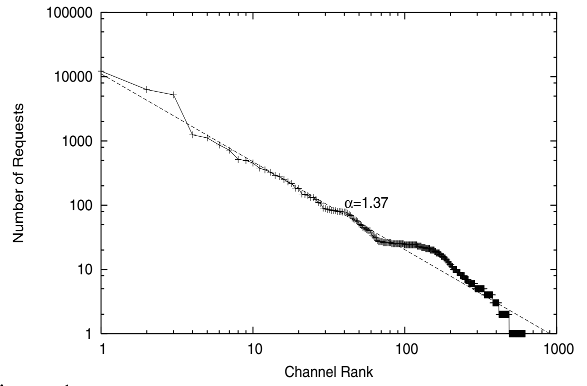


Figure 1: **Feeds Ranked by the Number of Requests: RSS popularity follows a Zipf distribution.**

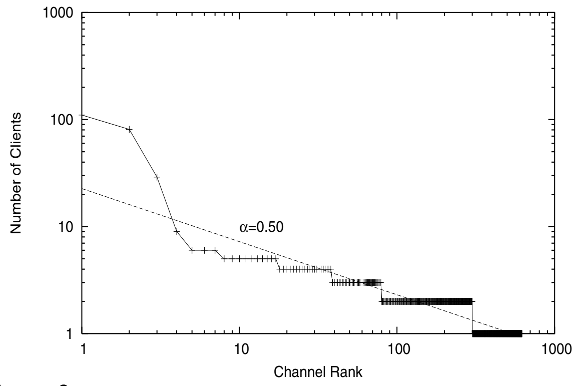


Figure 2: **Feeds Ranked by the Number of Subscribers: RSS popularity based on subscriptions also follows a Zipf distribution.**

May 2005, and recorded all RSS related traffic. Table 1 provides a summary of the trace.

Our tracer software operates by capturing every TCP packet, reassembling full TCP flows, and logging the flows that contain an RSS request or response. For anonymity, we obfuscate client IP addresses using a one-way hash salted with a secret; this enables us to identify unique IP addresses without being able to map them back onto hosts. Although DHCP is used in our department, the assignment of IP addresses is decided by the physical network port used, and is therefore quite static. Laptop users that connect to public network ports may have different IPs over time, but we estimate the number of laptop users to be low compared to users with fixed IPs. The tracer tool ran on a Dell dual processor 4650 workstation, which was able to keep up with packet capture at Gigabit line speed on the link from our department to the campus backbone. We made flow assembly non-performance critical by performing it offline on the captured packet stream and observed no packet drops during the whole trace period.

Active Polling

We obtained a list of 99,714 RSS feeds from *syndic8.com*, a directory that acts as a vast repository of RSS feeds. We actively polled these feeds every hour for 84 hours and recorded the results. While fetching the feeds, download timeout was set to 20 seconds and a request was retried 4 times if the response was not received within the timeout period. A successful download of the RSS content gives a snapshot of the RSS feed at that time. A download may fail due to high instantaneous load on the server, network congestion, or stringent polling limits imposed by servers. We fetched 3,682,043 snapshots in total; that is, about 36.9 snapshots per feed on average. The results of active polling are summarized in Table 2.

4 Survey Results

We report on three broad aspects of the RSS system using the trace data and periodic snapshots. First, we analyze the characteristics of RSS feeds, such as the popularity distribution, content size, format, and version of RSS used. Second, we investigate how RSS feeds are updated; in particular, we focus on the update intervals of RSS feeds, the amount of change involved in updates, and correlations between updates and feed size. Finally, we examine how clients use RSS by studying their polling behavior and subscription patterns. This section describes our findings in detail.

4.1 Feed Characteristics

We first present statistics on RSS workload and content. We compute the popularity of RSS feeds based on the user activity traces and derive content characteristics from the snapshots of RSS feeds. We measure popularity in two ways: based on the number of requests received for each RSS feed and based on the number of clients who subscribed to each RSS feed.

Feed Popularity

Figure 1 shows the popularity of RSS feeds ranked by the number of requests received. The popularity follows roughly a Zipf (power law) distribution with α parameter 1.37. The most popular feed (BBC news) receives 12,203 requests, while there is a long tail of many feeds that receive only a single request. Figure 2 plots the popularity of RSS feeds based on number of subscribers observed in the trace. The distribution of subscribers also follows a Zipf distribution (α = 0.5). The small number of clients in our trace makes the log-log plot diverge a little from the Zipf line. Overall, RSS workload has characteristics similar to Web workload, which is also known to follow heavy-tailed power-law distributions [3].

Feed Format and Version

We find that RSS is the widely-used format with more than 98% (97720 feeds) of the feeds in RSS; a small 2% (1994) of the feeds, however, use Atom [2], another XML based format for disseminating Web micronews. We further breakdown the RSS feeds according to their versions and show the results in Figure 3. Version 2.0 is the most popular format; more than 60% of RSS feeds published on *syndic8.com* are in this format. Version 0.91 and 1.0 count for about 17% each. Other versions(0.90, 0.93 and 0.94)

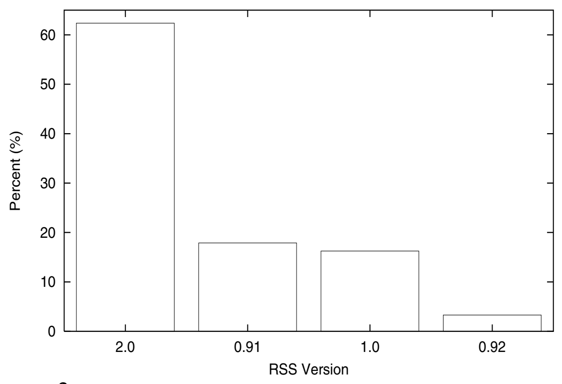


Figure 3: **Distribution of RSS Version. Version 2.0 is the most popular version of RSS.**

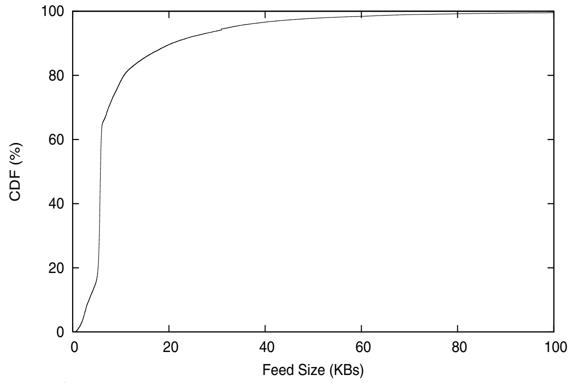


Figure 4: **CDF of Feed Size: RSS feeds are typically small (less than 10 KB) with a median of about 5.8 KB.**

are rare; they count for only 0.2% in total, and therefore are not shown in the figure.

Feed Size

RSS feeds typically consist of Web content encapsulated in XML format. Therefore, we expect the majority of RSS feeds to have size close to most Web objects. This is confirmed by Figure 4, which plots the distribution of feed size. The feed size is calculated as the average of all the snapshots of the feed; the variance is very small for the feed snapshots. More than 80% of the RSS feeds are relatively small at less than 10KB. The minimum observed feed size is 356 bytes, median is 5.8KB, and the average is 10KB. While, 99.9% of feeds are smaller than 100KB, the feed size distribution is heavy tailed with the largest feed at 876,836 bytes (not shown in the graph).

Extremely large RSS feeds, however, are rare, unlike some Web objects that can be of several megabytes or more. The concise nature of RSS feeds is expected because RSS is meant for the quick dissemination of news updates, often only carrying links to the more elaborate news articles. Moreover, the current architecture of RSS, where clients need to fetch the whole feed for checking updates, poses a high bandwidth load on content servers. This discourages content providers from supporting large feeds and biases towards small feed sizes.

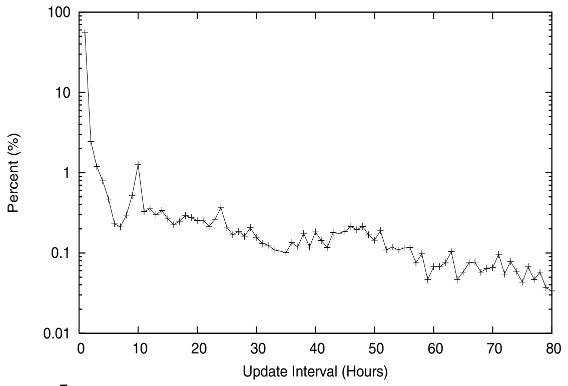


Figure 5: **Distribution of First Update Intervals: 55% of feeds get updated in an hour.**

4.2 Update Characteristics

Updates are the main driving force of the RSS pub-sub system. We examine the nature of RSS updates using the series of hourly snapshots gathered through active polling. We ensure that missing snapshots do not affect the calculations of update interval by only counting the intervals between *valid updates*; an update is valid only if there is a valid snapshot preceding the update, and that preceding snapshot matches the last recorded update. In order to calculate the update characteristics accurately, we filtered out all the feeds that have less than thirty snapshots leaving 68,266 feeds.

Update Rate

Figure 5 shows the distribution of update intervals of the first valid update. We see that feed update rates fall in two extremes: they either update very frequently or very rarely. More than 55% of feeds are updated in the first hour, while 25% of feeds did not see any updates during the entire polling period. This result suggests that RSS readers should use different polling periods for different feeds. However, some RSS readers, e.g., Thunderbird 1.0, do not support this feature currently.

Figure 6 shows the average update interval of RSS feeds, calculated by averaging the valid update intervals measured for each feed. We see that over 57% of the RSS feeds have an average update interval under two hours. Since we gathered snapshots by the hour, our data do not show updates that happen within an hour. Nevertheless, we find that RSS feeds often change at a rapid rate and RSS readers need to poll aggressively in order to detect updates quickly.

Update Size

We quantify update sizes using the minimum edit distance ("diff") between two consecutive snapshots. Figure 7 shows the cumulative distribution of update sizes. 64% of all updates involve no more than two lines of changes. The average change in the number of lines is 16.7 (6.8% of feed size) and the maximum is 16,542. The feed that changes most is hosted by a weather service website that provides weather forecast for many areas.

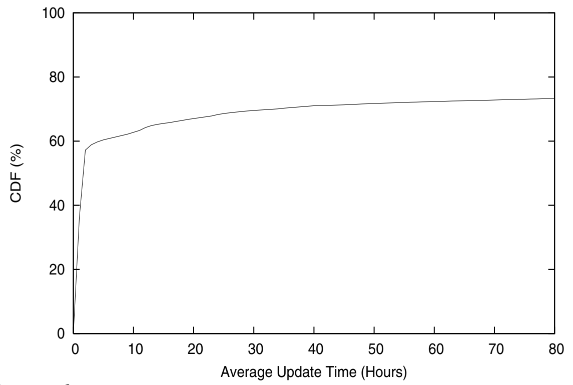


Figure 6: **Average Update Time: 57% of feeds have average update interval of less than two hours, while 25% of feeds do not change for more than three days.**

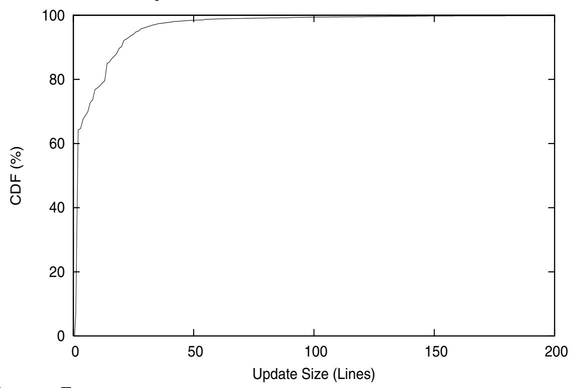


Figure 7: **Number of Changed Lines in Updates: 64% of updates involve no more than two lines of change.**

The major criticism against RSS has centered around its scalability. The constant polling by clients poses a significant bandwidth challenge on RSS servers. There have been many proposals for reducing the bandwidth consumption. For instance, RSS 2.0 supports the *TTL*, *SkipDays*, and *SkipHours* tags to advise the clients to choose an optimal polling rate and to skip periods when no updates are available, such as weekends. But a better solution is to send clients only the "delta," that is, the portion of data that actually changes. Our measurement shows that the feed updates only 6.8% of its content on average, which suggests that this optimization can reduce bandwidth consumption by as much as 93.2%.

Correlations between Feed Size and Updates

We explore the correlation between feed size and update rates and sizes. Figure 8 shows the average number of updates as a function of feed size. Though the data indicates some peaks, there is no strong correlation between size and update rate. We suspect that the peaks are due to commonly used, frequently changing XML objects clustered around certain sizes. However, there is a correlation between feed size and update size, as can be seen in Figure 9. For most feeds, the average update size grows as feed size increases. For feeds smaller than 68KB (about 99% of the total), the correlation coefficient is 0.89. The curve becomes irregular

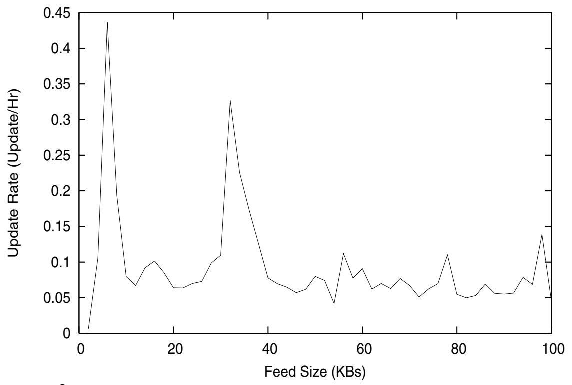


Figure 8: **Correlation Between Feed Size and Update Rate: There is no noticeable correlation between feed size and update rate.**

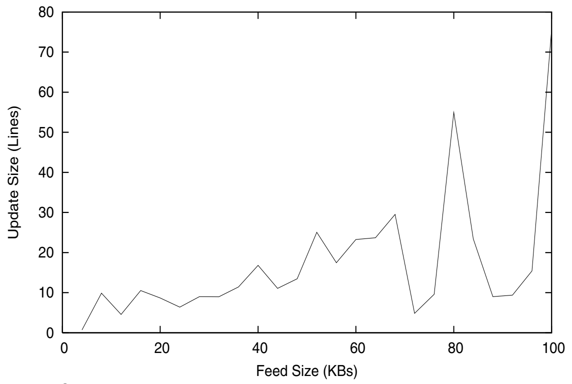


Figure 9: **Correlation between Feed Size and Update Size: The amount of change during an update increases with the feed size.**

after feed size increases more than 68KB due to the small number of samples available.

4.3 Client Behavior

Finally, we analyze how clients use the RSS system from the user activity trace we collected.

Polling Frequency

We divide the clients into two categories, namely *auto and manual*, according to their polling behavior. Auto clients poll feeds at a fixed rate, usually by running RSS readers in the background, while manual clients use RSS in the same way as they browse the Web, that is, launch RSS readers when they really want to read the news, and close the program after reading it. We consider clients who poll a feed for less than 3 times a day or with irregular polling intervals as manual clients. We find that 36% of clients in our department fall in this category. For auto clients, who poll at periodic intervals, we show the polling rate in Figure 10. 58% of them poll feeds hourly, suggesting that most users are "lazy", and do not change the default setting of their RSS readers. A small number of aggressive clients poll as often as every ten minutes.

Number of Subscriptions

Figure 11 shows the number of feeds subscribed by each client in sorted order. This distribution also follows a Zipf distribution with α parameter around 1.13. While most

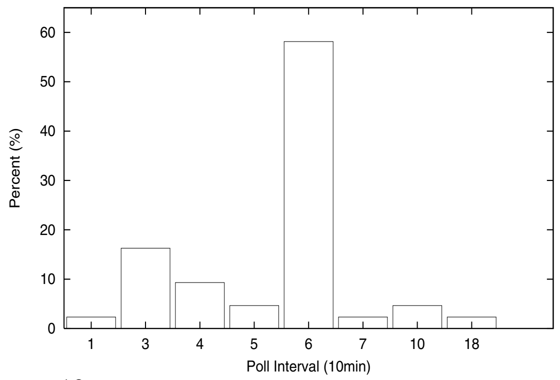


Figure 10: **Polling Rate of Clients: About 58% of clients use the default setting of one hour as the polling period.**

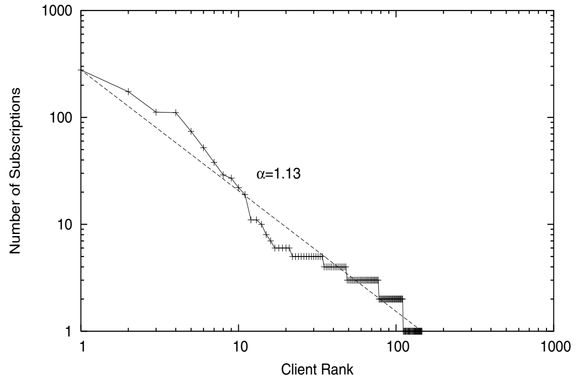


Figure 11: **Number of Subscriptions made by Clients: The number of channels subscribed by clients follows a Zipf distribution.**

clients subscribe to less than five feeds, there are several clients that subscribe to more than 100 feeds.

5 Discussions and Conclusions

This paper presents a measurement study of RSS, a pub-sub system for disseminating Web micronews. It provides insights about how a pub-sub system is utilized in practice and what issues need to be addressed while designing pub-sub systems.

The main focus of our study is to analyze how feeds are updated, a fundamental aspect of pub-sub systems. This study shows that update rates of RSS feeds are distributed in extremes; a majority of feeds (55%) update every hour, while many feeds (25%) do not change for days together. Hence, significant bandwidth savings can be obtained by using the optimal polling period for each feed instead of a single common polling rate for all feeds. End users of RSS, however, cannot be relied on to set the optimal polling rate, as this study shows that clients predominantly do not change the default settings of RSS readers. A better solution is for content providers to indicate when and at what rate to poll a particular feed. The version 2.0 of RSS already provides support for customized polling, although many readers are yet to support this feature.

Much of the bandwidth in RSS goes towards refetching feeds in order to check for updates because the current RSS architecture does not employ asynchronous notifications. This study indicates that delta encoding is a major opportunity for improving bandwidth usage in RSS, as updates are often made only to a tiny portion of the content (about 7% of the feed on average). Moreover, clients subscribed to the same feed poll the content servers independently, imposing a high load on the servers of popular feeds. Recently proposed systems [11, 10] use peer-to-peer overlays for cooperative polling to alleviate load on the servers and to provide faster updates. Such systems capable of asynchronous update notifications seem to be a step in the right direction.

Overall, this is the first study of a widely deployed pub-sub system performed during the early days of RSS. We hope this study will help to understand, design, and evaluate future pub-sub systems, and more studies with greater depth will emerge as the popularity of RSS increases.

References

[1] Air Force Research Laboratory (AFRL/IF) JBI Team. Joint Battlespace Infosphere. *http://www.rl.af.mil/programs/jbi/*, 2005.

[2] Atom Enabled. Atom Syndication Format. *http://www.atomenabled.org/developers/syndication*.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. of IEEE International Conference on Computer Communications*, New York, NY, Mar. 1999.

[4] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a Global Event Notification Service. In *Proc. of the Workshop on Hot Topics in Operating Systems*, Elmau, Germany, May 2001.

[5] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, Apr. 1989.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[7] F. Douglis, A. Feldman, B. Krishnamurthy, and J. Mogul. Rate of Change and Other Metrics: a Live Study of the World Wide Web. In *Proc. of USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, Dec. 1997.

[8] B. Glade, R. Cooper, R. van Renesse, and K. Birman. Light-Weight Process Groups in the ISIS System. *Distributed Systems Engineering*, 1(1):29–36, Sept. 1993.

[9] IBM. TSpaces - Computer Science Research at Almaden. *http://www.almaden.ibm.com/cs/TSpaces/*.

[10] V. Ramasubramanian, R. N. Murthy, and E. G. Sirer. Corona: A High Performance Publish-Subscribe System for Web Micronews. *http://www.cs.cornell.edu/people/egs/beehive/corona*.

[11] D. Sandler, A. Mislove, A. Post, and P. Druschel. FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification. In *Proc. of International Workshop on Peer-to-Peer Systems*, Ithaca, NY, Mar. 2005.

[12] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proc. of International Symposium on Software Reliability Engineering*, Paderborn, Germany, Nov. 1998.

[13] TIBCO. TIBCO Publish-Subscribe. *http://www.tibco.com/software/*.

[14] UserLand. RSS 2.0 Specifications. *http://blogs.law.harvard.edu/tech/rss*, 2005.

[15] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

Measurements, Analysis, and Modeling of BitTorrent-like Systems

Lei Guo[1], Songqing Chen[2], Zhen Xiao[3], Enhua Tan[1], Xiaoning Ding[1], and Xiaodong Zhang[1]

[1]*Department of Computer Science*
College of William and Mary
Williamsburg, VA 23187, USA
{lguo, etan, dingxn, zhang}@cs.wm.edu

[2]*Department of Computer Science*
George Mason University
Fairfax, VA 22030, USA
sqchen@cs.gmu.edu

[3]*AT&T Labs-Research*
180 Park Ave.
Florham Park, NJ 07932, USA
xiao@research.att.com

Abstract

Existing studies on BitTorrent systems are single-torrent based, while more than 85% of all peers participate in multiple torrents according to our trace analysis. In addition, these studies are not sufficiently insightful and accurate even for single-torrent models, due to some unrealistic assumptions. Our analysis of representative BitTorrent traffic provides several new findings regarding the limitations of BitTorrent systems: (1) Due to the exponentially decreasing peer arrival rate in reality, service availability in such systems becomes poor quickly, after which it is difficult for the file to be located and downloaded. (2) Client performance in the BitTorrent-like systems is unstable, and fluctuates widely with the peer population. (3) Existing systems could provide unfair services to peers, where peers with high downloading speed tend to download more and upload less. In this paper, we study these limitations on torrent evolution in realistic environments. Motivated by the analysis and modeling results, we further build a graph based multi-torrent model to study inter-torrent collaboration. Our model quantitatively provides strong motivation for inter-torrent collaboration instead of directly stimulating seeds to stay longer. We also discuss a system design to show the feasibility of multi-torrent collaboration.

1 Introduction

BitTorrent [8] is a new generation of Peer-to-Peer (P2P) system that has become very popular recently. According to a recent CNN report, BitTorrent traffic represents 53% of all P2P traffic on the Internet in June 2004 [16]. Unlike traditional P2P systems such as Gnutella [1], KaZaa [2], and eDonkey/eMule/Overnet [3], in which peers sharing *different* files are organized together and exchange their desired files with each other, BitTorrent organizes peers sharing the *same* file into a P2P network and focuses on fast and efficient replication to distribute the file. In BitTorrent, a file is divided into small chunks, and a peer can download multiple chunks of the file in parallel. Peers with different file chunks are stimulated to exchange with each other through a "tit-for-tat" incentive mechanism, which enables peers with high uploading bandwidth to have corresponding high downloading bandwidth. In this way, BitTorrent prevents *free riding* effectively, which is very common in early P2P systems [5]. P2P systems for exchanging different files such as KaZaa and eMule use participation levels or credit/reputation systems to track the contribution of each peer, and encourage peers to contribute by giving higher service priority to those peers with more contribution. However, such systems are either too complex and unrealistic or very easy to be circumvented [4, 6]. Compared to these systems, the direct "tit-for-tat" mechanism of BitTorrent is very simple and effective. In practice, BitTorrent-like systems scale fairly well during flash crowd period and are now widely used for various purposes, such as for distributing large software packages [7, 14].

Research has been conducted to study the effectiveness of BitTorrent-like systems [7, 14, 17, 18, 23]. The most recent work shows the stability of BitTorrent-like systems through a fluid model, and verifies the effectiveness of the current incentive mechanism [18]. However, this fluid model assumes a Poisson arrival model for the requests, which has been shown to be unrealistic during a long period (eight months) of trace study [17]. Consequently, the model can only characterize the performance of the BitTorrent system under stable conditions. In reality, as shown by our trace analysis, the stable period is very short. In addition, all existing studies on BitTorrent-like systems focus on the behaviors of single-torrent systems only, while our traces show that most peers ($> 85\%$) participate in multiple torrents.

In this work, we present an extensive study of BitTorrent-like P2P systems through measurements, trace analysis, and modeling. We first study the evolution of BitTorrent systems based on realistic assumptions analyzed from traces. We find that although the existing system is effective to address the "flash crowd" problem upon the debut of a new file, it has the following limitations:

- Due to the exponentially decreasing peer arrival rate and the lack of seeds (peers with a full copy of the file), service availability in the BitTorrent system becomes poor quickly, after which it is difficult for the file to be located and downloaded.
- Client performance in the BitTorrent system is unstable and fluctuates substantially with peer population variations.
- Existing systems can provide unfair services to peers. Studying the peer contribution ratio (uploaded bytes over downloaded bytes), we find that the peer contribution ratio decreases with its downloading speed.

Motivated by the results for the single-torrent system, we further study the multi-torrent system through trace analysis and modeling. Although it was generally understood that collaboration among multiple torrents might overcome some of the limitations of the single-torrent system, to our best knowledge, our work is the first to quantitatively and comprehensively analyze the multi-torrent system. In detail, we (1) characterize the peer request pattern in multiple torrents; (2) study the service potentials a torrent can provide to and get from other torrents; (3) demonstrate the benefit of inter-torrent collaboration. In addition, we discuss a new architecture to facilitate inter-torrent collaboration and show the feasibility and compatibility to the current BitTorrent systems.

Our contributions in this work are:

- We find three limitations of existing BitTorrent-like systems through torrent evolution study based on correct peer arrival pattern.
- Motivated by the modeling and analysis results, we build a graph-based multi-torrent model to quantify the inter-torrent collaboration benefit. The result shows that inter-torrent collaboration is much more effective than directly stimulating seeds to stay longer, addressing the well-known problem of lacking incentives to seeds.
- Guided by the modeling result, we propose and discuss a new architecture for inter-torrent collaboration.

The remainder of the paper is organized as follows. Section 2 presents related work. We demonstrate the limitations of existing BitTorrent-like systems through measurements, trace analysis, and modeling in Section 3, and present our multi-torrent model in Section 4. Section 5 proposes and discusses an architecture for inter-torrent collaboration. We make concluding remarks in Section 6.

2 Other Related Work

The amount of P2P traffic and the population of P2P users on the Internet keeps increasing. A lot of studies have been performed on the measurements, modeling, and algorithms of different P2P systems.

Measurement studies [19, 20] characterize the P2P traffic over the Internet, including Napster, Gnutella, and KaZaa systems. Study [12] analyzes the popularity of P2P content over the Internet and characterizes the "download at most once" property of P2P clients. Extensive measurements and traffic analysis on BitTorrent systems have also been conducted recently. Study [14] analyzes a five-month workload of a single BitTorrent system for software distribution that involved thousands of peers, and assesses the performance of BitTorrent at the flash crowd period. In [7], authors analyze the BitTorrent traffic of thousands of torrents over a two-month period, with respect to file characteristics and client access characteristics. Work [17] presents the current infrastructure of BitTorrent file sharing systems, including the Web servers/mirrors for directory service, meta-data distribution, and P2P content sharing. The authors also find that the arrival, abort, and departure processes of downloaders do not follow a Poisson distribution in the eight-month trace they collected, which was assumed in the previous modeling study [18].

A queuing model for P2P file sharing systems is proposed in [11]. Study [23] analyzes the service capacity of BitTorrent-like systems, and finds that multi-part downloading helps P2P systems to improve performance during flash crowd period. Study [18] further characterizes the overall performance of BitTorrent-like systems using a simple fluid model, and analyzes the effectiveness of BitTorrent incentive mechanism using game theory. Study [15] introduces a probabilistic model of coupon replication systems, and analyzes the performance under an environment where neither altruistic user behaviors nor load balancing strategies (such as rarest first in BitTorrent) are supported.

Study [22] proposes an interest-based content location approach for P2P systems. By self-organizing into small groups, peers with the same interest can collaborate more efficiently, which is similar to the BitTorrent networks, where all peers share the same file. In [21], a P2P protocol is proposed for bulk data transfer, which aims to improve client performance and to reduce server load, by using enhanced algorithms over BitTorrent systems.

Different from all studies above, our modeling and trace analysis provide an understanding of torrent evolution in the BitTorrent systems and the relation among multiple torrents over the Internet. Furthermore, our results reveal three limitations in current BitTorrent systems, and propose an innovative architecture to facilitate inter-torrent collaboration, which represents the first step towards making the current BitTorrent-like system a reliable and efficient content delivery vehicle.

3 Modeling and Characterization of BitTorrent-like Systems

In a BitTorrent system, the content provider creates a *meta file* (with the `.torrent` suffix name) for the *torrent file* it wants to share, and publishes the meta file on a Web site. Then the content provider starts a BitTorrent client with a full copy of the torrent file as the original *seed*. For each torrent file, there is a *tracker site*, whose URL is encoded in the meta file, to help peers find each other to exchange the file chunks. A user starts a BitTorrent client as a *downloader* at the beginning to download file chunks from other peers or seeds in parallel. A peer that has downloaded the file completely also becomes a seed that could in turn provide downloading service to other peers. All peers in the system, including both downloaders and seeds, self-organize into a P2P network, known as a *torrent*. The initial seed can leave the torrent when there are other seeds available, and content availability and system performance in the future depend on the arrival and departure of downloaders and other seeds.

Previous research has studied BitTorrent-like systems through trace analysis and modeling, and verified its effectiveness during flash crowds, which normally happen soon upon the debut of a new file [18]. However, no existing work has characterized overall client performance in the lifetime of a torrent when the file popularity changes. This is particularly important for BitTorrent-like systems where service availability relies purely on the voluntary participation of peers. This is in contrast to a client-server model where a permanent site (i.e., a server) can provide persistent service.

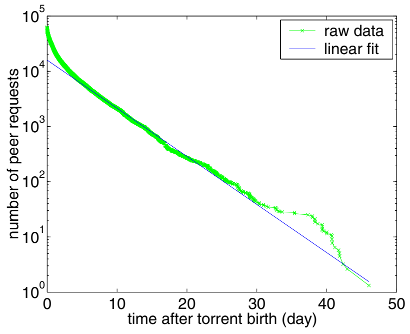
In this section, we study torrent evolution, downloading service availability, client performance fluctuation, and service fairness in BitTorrent-like systems based on torrent popularity characterization. We propose an evolution model for BitTorrent-like systems and analyze the torrent lifespan, ratio of failed peers, and the service policy of seeds, based on both the modeling and trace analysis.

3.1 Torrent Popularity Characterization

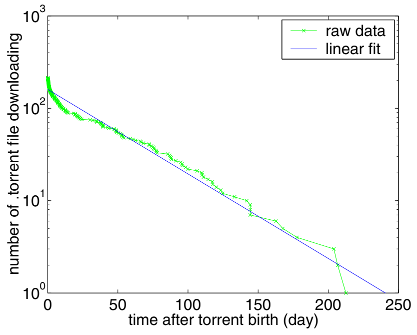
In this study, we analyze and model BitTorrent traffic based on two kinds of traces. The first one contains the statistics collected from two popular dedicated tracker sites (although each torrent can have its own tracker site, there are many dedicated tracker sites on the Internet providing persistent service, each of which may host thousands of torrents), sampled every half an hour for 48 days from 2003-10-23 to 2003-12-10. This trace was collected by University of Massachusetts, Amherst [7] (abbreviated as the *tracker trace* or *trace* in the remainder of this paper). We identify different peers and match multiple sessions of the same downloading with the similar methods used in study [14]. The firewalled peers, although they cannot accept incoming connections and thus are not listed in the tracker query responses to allow other peers to connect to, are still included in the tracker statistics. We extract the peer request time, downloading/uploading bytes, the downloading/uploading bandwidth of all peers for each torrent, and the information for each torrent such as torrent birth time and file size. Due to space limitations, we only present the analysis results of the larger tracker trace, which includes more than 1,500 torrents (about 550 torrents are fully traced during their lifecycles). The smaller trace has similar results.

In order to better understand BitTorrent traffic over the Internet, we also collected the BitTorrent meta file downloading trace from a large commercial server farm hosted by a major ISP and a large group of home users connected to the Internet via a well-known cable company, using the Gigascope appliance [10], from 2004-09-28 to 2004-10-07. The *server farm trace* includes about 50 tracker sites hosting hundreds of torrents, and the *cable network trace* includes about 3,000 BitTorrent users (by IP addresses) requesting thousands of torrents on the Internet. Both traces include the first IP packets of all HTTP downloading of the `.torrent` files, with the timestamp when the packet is captured (the downloading time of the `.torrent` file). This timestamp represents the peer arrival time to the torrent. We also extract the timestamp encoded in each `.torrent` file, which is the creation time of the meta file and represents the torrent birth time.

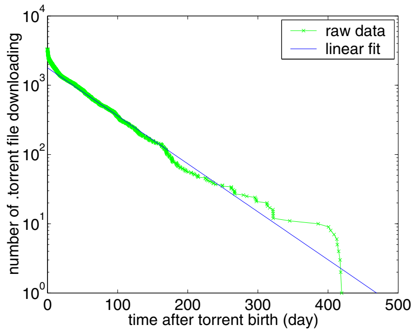
Figure 1(a) shows the complementary CDF (CCDF) distribution of the time after torrent birth for the requests to all fully-traced torrents in the tracker trace. For peers downloading the file in multiple sessions, only the first requests are considered. The y-axis at time t denotes the total number of requests for all torrents in the trace minus the cumulative number of requests for all torrents after time t since they are born. Figures 1(b) and 1(c) show the CCDF distribution of the time when a `.torrent` file was downloaded after torrent birth in the server farm



(a) Tracker trace



(b) Server farm trace



(c) Cable network trace

Figure 1: The complementary CDF distribution of peer arrival time (time of a peer's first request to a torrent or time when a meta file was downloaded) after torrent birth for three BitTorrent traces (y-axis is in log scale)

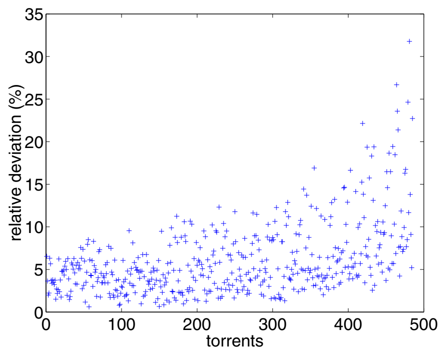


Figure 2: Fitting deviations of fully-traced torrents in the tracker trace

and in the cable network, respectively. Note that y-axis is in log scale in the three figures.

All three curves can be fitted with straight lines. This consistent trend strongly suggests that after a torrent is born, the number of peer arrivals to the torrent decreases exponentially with time in general. To validate that this conjecture holds for individual torrents as well, we use the least square method to fit the logarithm of the complementary of the number of peer arrivals for each torrent in the tracker trace. We define the *relative deviation* of the fitting for the number of requests at a time instant as $\frac{|\log N_0 - \log N|}{\log N_0} \times 100\%$, where N_0 is the actual complementary value of the number of requests and N is the fitting result. Figure 2 shows the distribution of average fitting deviation for each fully-traced torrent that has at least 20 peers during its lifetime. In this figure, each point in the x-axis denotes a torrent, sorted in non-ascending order of torrent population during the entire lifetime, and the corresponding value in y-axis denotes the average of relative fitting deviation of this torrent. We can see the fitting is more accurate for torrents with larger population, and the overall average relative deviation is only about 6%. We do not fit the curve for individual torrents in the server farm and cable network trace, because the data collection duration is short so that they do not cover the whole lifespans of torrents. In the remainder part of this paper, we only use the tracker trace for modeling and analysis.

We define the *torrent popularity* at a time instant as the peer arrival rate of the torrent at that time, which is the derivative of the peer arrival time distribution of that torrent. Since the derivative of an exponential function is also an exponential function, we assume that the peer arrival rate of a torrent follows an exponential decreasing rule with time t

$$\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}, \tag{3.1}$$

where λ_0 is the initial arrival rate when the torrent starts, and τ is the attenuation parameter of the torrent evolution. In Section 3.3, we will use a fluid model to evaluate our assumption again.

3.2 Evolution and Service Availability of BitTorrent

We define the *torrent lifespan* as the duration from the birth of the torrent to the time after which there is no complete copy of the file in the system, and the new arriving peers cannot complete downloading. To simplify the modeling, we assume that the initial seed exits the system as soon as a downloader has downloaded the file completely. In practice, the initial seed may stay online in the system for a longer time, and some seeds may return to the system to serve the content.

The *inter-arrival time* between two successive arriving peers δt can be approximated as $\frac{1}{\lambda}$. Denote the rate at which seeds leave the system as γ, then the average service time of a seed can be approximated as $\frac{1}{\gamma}$. As shown in Figure 3, peer n and peer n+1 are the n-th and (n+1)-th arriving peers in the torrent, at the time t_n and t_{n+1},

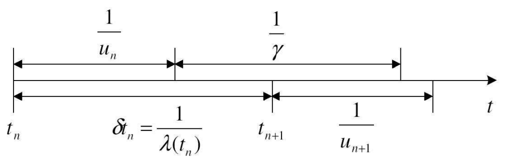


Figure 3: The death of a torrent due to large inter-arrival time of peers

respectively. The inter-arrival time between peer n and peer n+1 can be estimated as $\delta t_n = t_{n+1} - t_n \approx \frac{1}{\lambda(t_n)}$. Peer n downloads the file with speed u_n and then stays in the torrent for a time duration $\frac{1}{\gamma}$. Peer n+1 downloads at speed u_{n+1}. According to the exponential decrease of peer arrival rate, the inter-arrival time of peers will grow exponentially, and finally there will be only one seed at a time. When the peer arrival rate $\lambda(t)$ is small enough (n is large), peer n+1 can only be served by peer n, and we have $u_{n+1} \leq u_n$. Thus, when $\delta t_n \approx \frac{1}{\lambda(t_n)} > \frac{1}{\gamma}$, peer $n+1$ cannot complete downloading before peer n leaves, and the torrent is dead. Using Equation 3.1, we get the torrent lifespan

$$T_{life} = \tau \log(\frac{\lambda_0}{\gamma}). \tag{3.2}$$

Equation 3.2 shows the expectation of the real torrent lifespan. To verify Equation 3.2, we compute the initial peer arrival rate λ_0 and the torrent attenuation parameter τ for fully traced torrents in the tracker trace. From Equation 3.1, we have

$$\log \delta t = -\log \lambda_0 + \frac{t}{\tau}. \tag{3.3}$$

Both δt and t for each peer arrival can be extracted from the trace and we get $\log \lambda_0$ and $\frac{1}{\tau}$ using linear regression. We also compute the seed leaving rate γ as the the reciprocal of the average seed service time, which is extracted from the trace, too. Figure 4 shows the comparison of torrent lifespan computed from the tracker trace (indicated by *trace*) and that from the Equation 3.2 (indicated by *model*). In this figure, each point in x-axis denotes a torrent, while each point in y-axis denotes the measurement result or the modeling result of torrent lifespan. The torrents in the x-axis are sorted in non-ascending order of the modeling results of torrent lifespans. As shown in the figure, our model fits the real torrent lifespan very well. The average lifespan of torrents is about 8.89 days based on the trace analysis and 8.34 days based on our model. The lifespans of most torrents are between 30 - 300 hours, and there are only a small number of torrents with extremely short or extremely long lifespans.

The *total population* of a torrent (in the number of peers) is

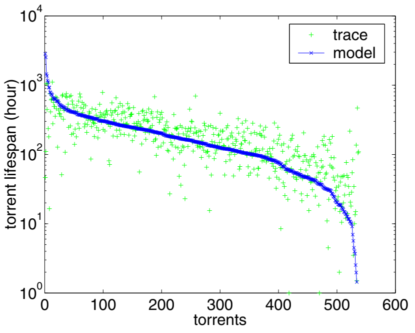


Figure 4: The comparison of torrent lifespan: modeling and trace analysis (y-axis is in log scale)

$$N_{all} = \int_0^\infty \lambda_0 e^{-\frac{t}{\tau}} dt = \lambda_0 \tau. \tag{3.4}$$

Among them, some peers may not be able to complete downloading due to lack of seeds, which we call *failed peers*, denoted as follows:

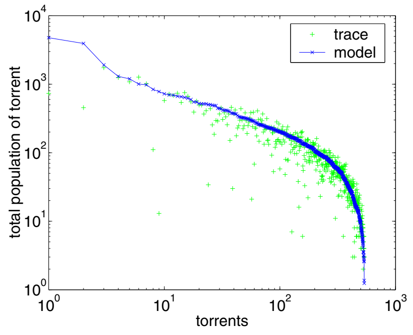
$$N_{fail} = \int_{T_{life}}^\infty \lambda_0 e^{-\frac{t}{\tau}} dt = \gamma \tau. \tag{3.5}$$

Thus, the *downloading failure ratio* of the torrent is

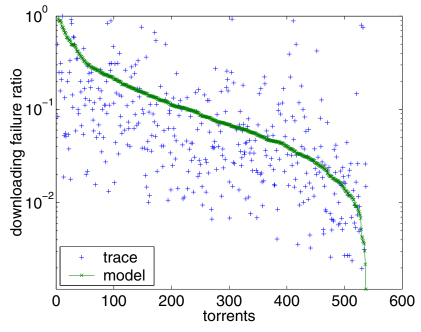
$$R_{fail} = \frac{N_{fail}}{N_{all}} = \frac{\gamma \tau}{\lambda_0 \tau} = \frac{\gamma}{\lambda_0}. \tag{3.6}$$

Figure 5(a) shows the comparison of the torrent population computed from the tracker trace with that from our model for each individual fully-traced torrent. In this figure, each point in x-axis denotes a torrent, while each point in y-axis denotes the measurement result or the modeling result of the total population of the torrent during its entire lifespan. The torrents in the x-axis are sorted in non-ascending order of the modeling results of torrent populations. As evidenced by the figure, the modeling result and trace analysis are consistent. In addition, we can see that the distribution of the torrent population is heavily skewed: although there are several large torrents, most torrents are very small, and the average population of torrents is only about 102 peers.

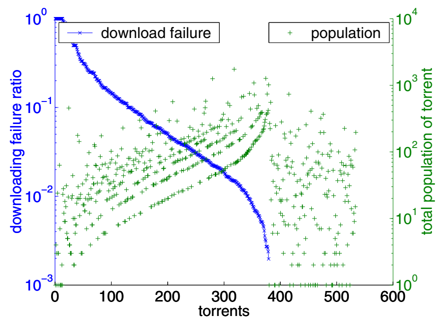
Figure 5(b) shows the downloading failure ratio based on trace analysis and on our model (plotted in the similar manner as that of Figure 5(a)). The real failure ratio of torrents is slightly lower than what our model predicts, because there are some altruistic peers that serve the torrent voluntarily. That also explains why the torrent lifespan in the trace analysis (8.89 days) is slightly higher than that in our model (8.34 days). Furthermore, there are some torrents that have no failed peers in the trace because the seeds leave after the downloaders finish, but



(a) The comparison of torrent population: modeling and trace analysis (in log-log scale)

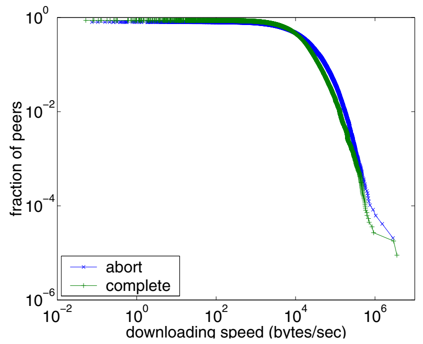


(b) The comparison of downloading failure ratio: modeling and trace analysis (y-axis is in log scale)

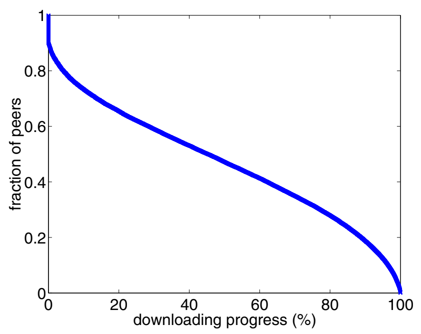


(c) The relation between torrent population and downloading failure ratio (y-axis is in log scale)

Figure 5: Torrent population and downloading failure ratio for all fully-traced torrents



(a) The downloading speed distribution (complementary CDF, in log-log scale)



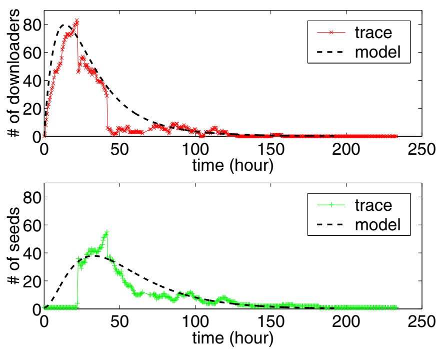
(b) The downloading progress distribution (complementary CDF)

Figure 6: The peers abort downloading voluntarily

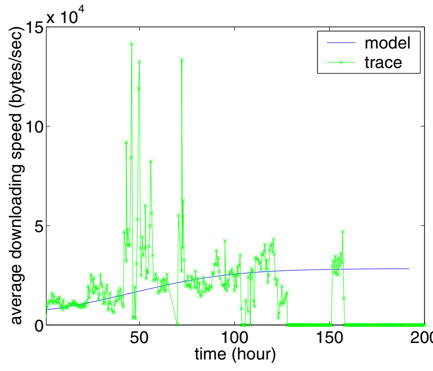
cannot be shown in the log scale plot. However, the average downloading failure ratio based on the trace analysis is still about 10%, which is non-trivial for a content distribution system.

Equation 3.5 implies that the number of failed peers in a torrent is independent of the initial peer arrival rate. Instead, the number of failed peers depends on the speed of torrent evolution (the attenuation exponent of peer arrival rate) and the seed departure rate. Figure 5(c) shows downloading failure ratios of torrents and their corresponding populations (plotted in the similar manner as that of Figure 5(a) and 5(b)). As reflected in the figure and indicated by Equation 3.5, the larger the torrent population, the lower the downloading failure ratio. It is interesting to note that the population of torrents, sorted in non-ascending order of their corresponding downloading failure ratios, forms several clear curves, each of which represents those torrents with similar evolution patterns (the popularity attenuation parameter τ). On the right side of the figure, the failure ratio of the torrents is 0 due to the existence of some altruistic seeds, which always stay until the last downloader completes.

In the above analysis, we assume that peers always complete their downloading unless they cannot. We do not consider peers that abort downloading voluntarily when seeds are still available in the torrent. A peer may abort downloading due to (1) loss of interest to the torrent file; (2) slow downloading speed or small downloading progress. Figure 6(a) shows the distribution of the average downloading speed of peers that voluntarily abort and peers that download the file completely. Figure 6(b) shows the distribution of downloading progress (the percentage of the entire file that has been downloaded) when peers abort downloading voluntarily. The figures indicate that the probability for a peer to abort downloading voluntarily is almost independent of its downloading speed and the current downloading progress. Hence, the voluntary aborting of some downloaders does not affect our analysis above.



(a) Torrent evolution



(b) Downloading speed

Figure 7: Torrent evolution under the fluid model

$x(t)$	number of downloaders in the system at time t
$y(t)$	number of seeds in the system at time t
λ_0	the initial value of peer arrival rate
τ	the attenuation parameter of peer arrival rate
μ	the uploading bandwidth
c	the downloading bandwidth ($c \gg \mu$)
γ	the rate at which seeds leave the system
η	the file sharing efficiency, meaning the probability that a peer can exchange chunks with other peers

Table 1: Notations and assumptions for the fluid model

3.3 Client Performance Variations in BitTorrent

Study [18] proposed a fluid model for BitTorrent-like systems with constant peer arrival rate. We follow the idea of the fluid model, but using the evolution of peer arrival rate described in Equation 3.1. The basic ODE (ordinary differential equation) set for the fluid model is

$$\begin{cases} \frac{dx(t)}{dt} = \lambda_0 e^{-\frac{t}{\tau}} - \mu(\eta x(t) + y(t)), \\ \frac{dy(t)}{dt} = \mu(\eta x(t) + y(t)) - \gamma y(t), \\ x(0) = 0, y(0) = 1, \end{cases} \tag{3.7}$$

where the meanings of the parameters in our fluid model are listed in Table 1. These notations are adopted from work [18, 23].

When the ODE set has two different real eigenvalues $\psi_1 \neq \psi_2$, the resolution can be expressed as:

$$\begin{cases} x(t) = ae^{\psi_1 t} + be^{\psi_2 t} + d_1 e^{-\frac{t}{\tau}}, \\ y(t) = c_1 ae^{\psi_1 t} + c_2 be^{\psi_2 t} + d_2 e^{-\frac{t}{\tau}}, \end{cases} \tag{3.8}$$

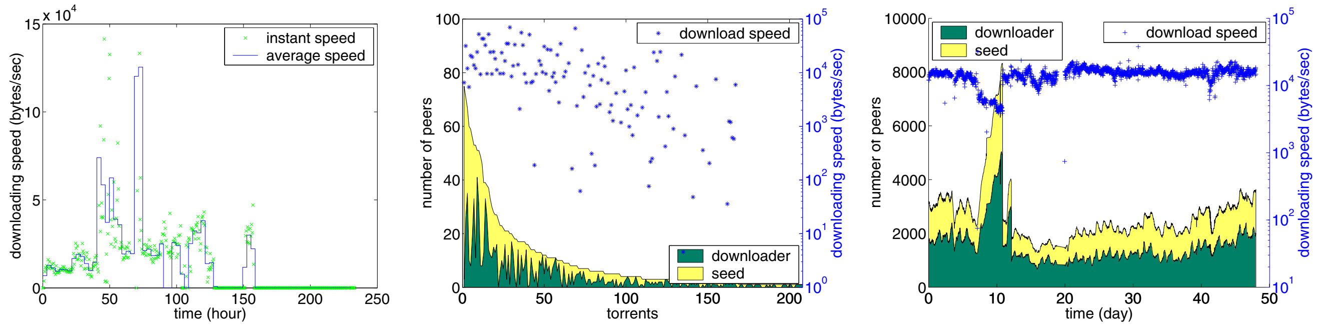
where d_1, d_2, c_1, c_2, a, b are constant. The value of these constants and the detailed resolution of the fluid model can be found in our technical report [13].

The average downloading speed of peers at time t is

$$u(t) = \mu\frac{\eta x(t) + y(t)}{x(t)} = \mu(\eta + \frac{y(t)}{x(t)}). \tag{3.9}$$

We use the tracker trace to validate the torrent evolution model. Similar to the peer arrival rate, the modeling results fit the trace better for torrents with larger populations. Figure 7(a) shows the torrent evolution by both our fluid model and the analysis results of a typical torrent in the trace. The figure shows that the number of downloaders increases exponentially in a short period of time after the torrent's birth (the flash crowd period), and then decreases exponentially, but at a slower rate. The number of seeds also increases exponentially at first, and then decreases exponentially at a slower rate. The peak time of the number of seeds lags behind that of the number of downloaders. As a result, $u(t)$ increases until the torrent is dead, and the resources of seeds cannot increase in proportion to service demand. Furthermore, due to the random arrival of downloaders and the random departure of seeds, average downloading performance fluctuates significantly when the number of peers in the torrent is small, as shown in Figure 7(b).

Figure 8(a) shows the performance variations of the torrent under two kinds of granularities. The *instant speed* represents the mean downloading speed of all peers in the torrent at that time instant, sampled every half an hour. The *average speed* represents the average value of the instant speed over the typical downloading time (the average downloading time of all peers). The figure shows that the client downloading speed at different time stages is highly diverse and can affect client downloading time significantly. The reason is that seeds play an important role in the client downloading performance. However, the generation of seeds is the same as the completeness of peer downloading, so the random fluctuation of downloading speed cannot be smoothed in

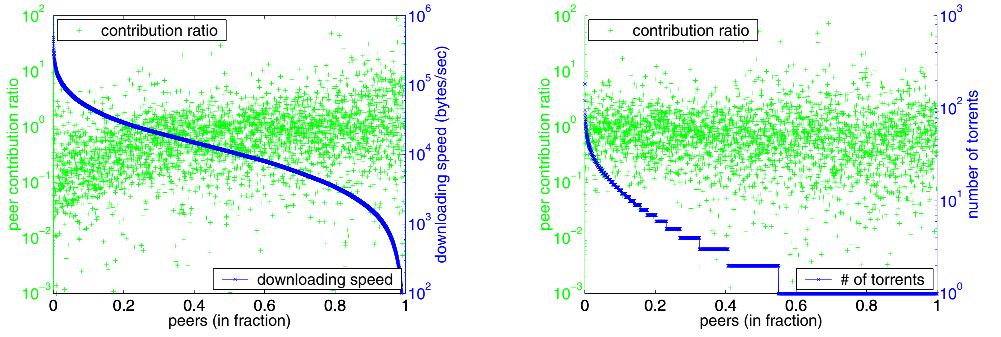


(a) The downloading speed in the lifetime of a typical torrent

(b) The downloading speed (in log scale) and the number of downloaders/seeds for each torrent at a time instant

(c) The average downloading speed (in log scale) and the total number of downloaders/seeds for all torrents in the system

Figure 8: Performance variations in BitTorrent systems



(a) The peer downloading speed and contribution ratio

(b) The the number of torrents that each peer involves and the corresponding contribution ratio

Figure 9: Fairness of seed service policy in BitTorrent systems (y-axis is in log scale)

the scale of typical downloading time when the number of peers is small.

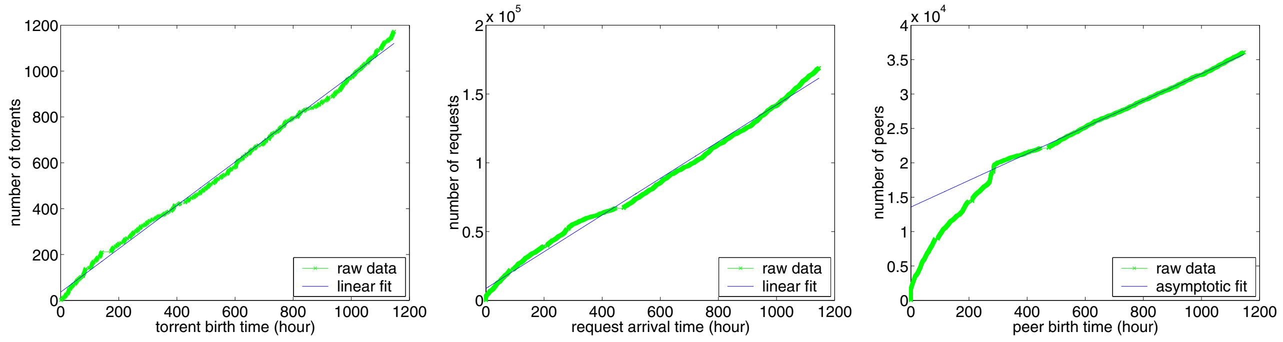
Figure 8(b) shows the number of downloaders and seeds (a stack figure), and the average downloading speed for each torrent in the trace at 12:00:01 on 2003-11-15. In this figure, each point in x-axis denotes a torrent, while the left y-axis denotes the number of downloaders and seeds in this torrent (stacked), and the right y-axis denotes average downloading speed of this torrent. The torrents in the x-axis are sorted in non-ascending order of the number of downloaders and seeds of torrents. The results at other time instants are similar. In general, peers in torrents with larger population have relatively higher and more stable downloading speed, while the downloading speed in torrents with small populations disperses significantly. When the number of peers in the torrent is small, the client downloading performance is easily affected by the individual behavior of seeds.

Figure 8(c) shows the total number of peers in all torrents (a stack figure) and the average downloading speed of all downloaders in the trace at different time stages. The average downloading speed of all torrents is shown to be much more stable than that of one torrent. The reason is that the downloader/seed ratio is much more stable due to the large population of the system. This motivates us to balance the service load among different torrents, so that each torrent can provide relatively stable downloading performance to clients in its lifespan.

3.4 Service Fairness Study in BitTorrent

In a BitTorrent system, the service policy of seeds favors peers with high downloading speed, in order to improve the seed production rate in the system, i.e., to have these high speed downloaders complete downloading as soon as possible and *wish* they will then serve other downloaders. In this subsection, we investigate whether this wish comes true in practice.

We define the *contribution ratio* of a peer as the total uploaded bytes over the total downloaded bytes of the peer. Figure 9(a) shows the peer downloading speed and the corresponding contribution ratio extracted from the trace. In this figure, each point in the x-axis denotes a



(a) Torrent birth (b) Request arrival of all peers over all torrents (c) Peer birth

Figure 10: The CDFs of torrent birth, peer request arrival, and peer birth over the trace collection time

peer, while the left y-axis denotes the contribution ratio of this peer, and the right y-axis denotes the average downloading speed of this peer. On the x-axis, peers are sorted in non-ascending order of their contribution ratios. The figure shows the rough trend that the peer contribution ratio increases when the downloading speed decreases. That is, the higher the downloading performance peers have, the less uploading service they actually contribute. This indicates that peers with high speed finish downloading quickly and then quit the system soon, which defeats the design purpose of the seed service policy.

Figure 9(b) shows the number of torrents that each peer involves and its corresponding contribution ratio (plotted in the similar way as that of Figure 9(a)). The figure shows no distinguishable correlation between the two, indicating that the main reason for seeds to leave old torrents is not to start new downloading tasks.

In summary, we observe that the BitTorrent's biased seed service policy in favor of high speed downloaders really affects the fairness to peers in downloading, and an incentive mechanism is needed to encourage seeds to contribute.

4 Modeling Multiple Torrents in BitTorrent Systems

In the previous section, we have shown that client performance fluctuates significantly in single-torrent systems, but is very stable when aggregated over multiple torrents. Based on this observation, in this section, we study the correlation among multiple torrents through modeling and trace analysis, aiming to look for solutions to enable inter-torrent collaboration.

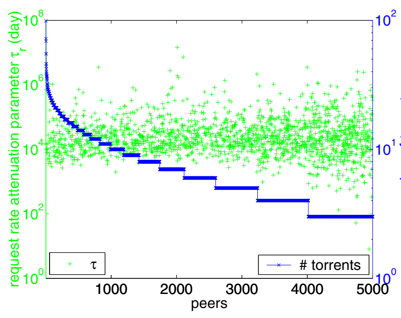
Although different torrents are independent from each other in the current BitTorrent systems, they are inherently related by peers that request multiple torrent files. A peer may download a torrent file, serve as a seed for that torrent for a while, and then go offline to *sleep* for some period of time. The peer may return sometime later and repeat the activities above. Thus, a peer's lifecyle consists of a sequence of *downloading*, *seeding*, and *sleeping* activities. If a peer stops using BitTorrent for a long time that is much longer than its typical sleeping time, we consider the peer as *dead*.

In the current BitTorrent systems, a peer is encouraged to exchange file chunks with other peers that are downloading the same file instead of serving old torrent files it has downloaded. Thus, in our model, we assume each peer joins (downloading and seeding) each torrent at most once, and joins one torrent at a time. Having these assumptions, we start to characterize peers in multiple torrents.

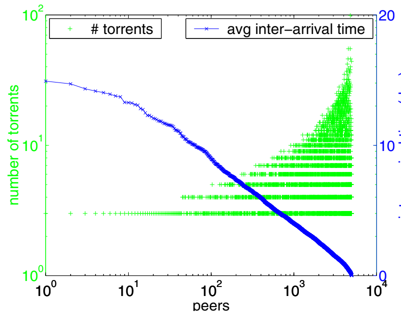
4.1 Characterizing the Peer Request Pattern

In the multi-torrent environment, both torrents and peers are born and die continuously. Figure 10(a) shows the CDF of torrent birth in the trace (indicated by *raw data*) and our linear fit. The average *torrent birth rate* (denoted as λ_t in the following context) is about 0.9454 torrent per hour. Figure 10(b) shows the CDF of torrent request arrivals (for all peers over all torrents) and our linear fit. We define the *torrent request rate* as the number of downloading requests for all torrents per unit time in the multi-torrent system, denoted as λ_q in the following context. Although the peer arrival rate of a single-torrent system decreases exponentially as shown in Figure 1, the torrent request rate in the multi-torrent system is almost a constant, about 133.39 per hour.

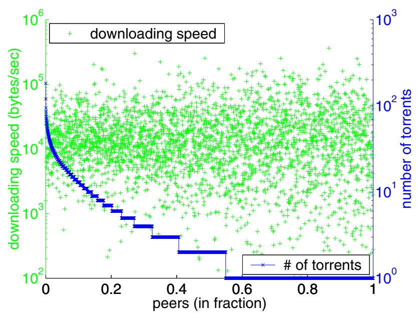
Since both the torrent birth rate and torrent request rate are almost constant, it is natural to assume that the *peer birth rate* (denoted as λ_p in the following context) is also a constant. A peer is *born* when it appears in the system for the first time. However, as shown in Figure 10(c),



(a) The attenuation of peers' requesting rates and number of torrents peers request (y-axis is in log scale)



(b) The inter-arrival time of peers' requests and number of torrents they join (in log-log scale)



(c) The downloading speed and number of torrents peers join (y-axis is in log scale)

Figure 11: The request pattern of peers

the peer birth rate is high at the beginning of the trace collection duration, and then converges to a constant rate asymptotically. The reason is that peers appear in the trace for the first time may actually be born before the trace collection, and the number of such peers decreases quickly after the trace collection starts. Thus, we take the asymptotic birth rate as the real birth rate of peers, which is about 19.37 per hour.

The constant peer birth rate and torrent request rate indicate that each peer only joins a limited number of torrents. However, the request rate of a peer might still change over time. We define the *peer request rate* as the number of requests a peer submits for different torrents per unit time. Assume the peer request rate can be expressed as

$$r(t) = r_0 e^{-\frac{t}{\tau_r}}, \tag{4.10}$$

where t is the time duration after the peer is born, r_0 is the initial request rate, and τ_r is the attenuation parameter of the request rate. When $\tau_r \rightarrow \infty$, the peer has a constant request rate; when $\tau_r < 0$, the peer has an increasing request rate.

The inter-arrival time between two successive requests of a peer δt is $\frac{1}{r(t)}$. Thus, we have

$$\log \delta t = -\log r_0 + \frac{t}{\tau_r}. \tag{4.11}$$

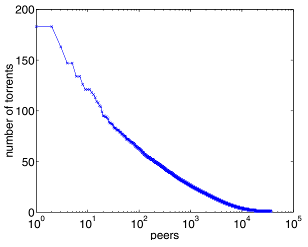
We extract δt and t from the trace for each peer requesting multiple torrents, and use linear regression to compute $\log r_0$ and $\frac{1}{\tau_r}$. Figure 11(a) shows the number of torrents that each peer requests and the corresponding τ_r. In this figure, each point in the x-axis denotes a peer, while the left y-axis denotes the τ_r value of this peer, and the right y-axis denotes the number of torrents this peer participates. In x-axis, peers are sorted in non-ascending order of the number of torrents they join. As shown in the figure, the value of parameter τ_r in Equation 4.10 is very large, with the mean value of about 77 years, which implies that the average request rates of peers do not change significantly over time. Further, τ_r is independent of the number of torrents that peers join. Thus, we can assume that the request processes of peers are Poisson-like processes with constant average request rates.

Figure 11(b) shows the average inter-arrival time of torrent requests for peers requesting multiple torrent files (plotted in the similar manner as that of Figure 11(a)). As shown in the figure, it is intuitive to find that the upper bound of the number of torrents each peer requests increases with the decrease of inter-arrival time. However, for peers with similar request rates, the number of torrents they request are very diverse, since they stay in the system for different time durations. Figure 11(c) further plots the downloading speed versus the number of torrents peers join (plotted in the similar manner as that of Figure 11(a)). There is no strong correlation between the two for peers with downloading speed $>$ 1 KB per second. This implies that for peers whose downloading speed is large enough, the numbers of torrent files different peers request do not depend on their request rates and their downloading speed.

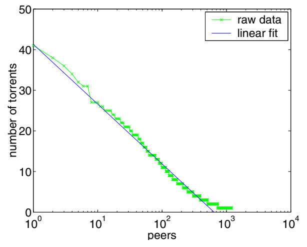
Thus, we assume that a peer joins a new torrent with probability p. For N peers in the system, during their whole lifecycles, there are Np^{m-1} peers that request at least m torrents. Ranking peers in non-ascending order of the number of torrents they join, the number of torrents that a peer ranked i joins is

$$m = 1 + \frac{\log i - \log N}{\log p}. \tag{4.12}$$

In addition, a peer has the probability $1-p$ to download exactly 1 file, probability $p(1-p)$ to download exactly 2 files, and probability $p^{k-1}(1-p)$ to download exactly k files. So the mean number of torrents that a

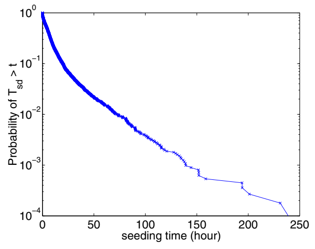


(a) For all peers in the trace

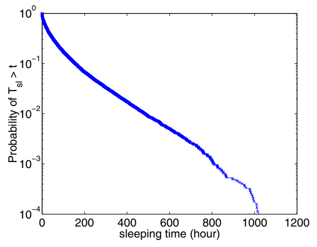


(b) For peers born in the middle of trace collection time

Figure 12: Torrent involvement of peers (x-axis is in log scale)



(a) The probability distribution of seeding time



(b) The probability distribution of sleeping time

Figure 13: The seeding time and sleeping time of peers (y-axis is in log scale)

peer joins is:

$$\bar{m} = \sum_{k=1}^{\infty} kp^{k-1}(1-p) = \frac{1}{1-p}. \tag{4.13}$$

Figure 12(a) shows the distribution of number of torrent files that each peer downloads in the trace. The curve in the figure is a little convex, deviating from what Equation 4.12 predicts (a straight line when x-axis is in log scale). The reason is that the number of torrents joined by peers born before the trace collection is under-estimated, since some of these requests cannot be recorded in the trace. A similar situation exists for peers that are not dead before the end of trace collection.

Figure 12(b) shows the distribution of number of torrents joined by each peer that was born in the middle of the trace collection duration (indicated by *raw data*) and our linear fit. The curve fits Equation 4.12 very well, and we estimate from the analysis that $p \approx 0.8551$, while the average number of torrents each peer joins is about 7.514.

To verify the probability model we use in the above analysis, we estimate p in another way as follows. Assuming the peer birth rate is λ_p and the torrent request rate is λ_q, since each peer joins $\frac{1}{1-p}$ torrents during its lifetime in average, we have

$$\lambda_q = \frac{1}{1-p}\lambda_p. \tag{4.14}$$

Based on the peer request arrival rate and the peer birth rate we derived before (see Figure 10(b) and 10(c)), we have $p = 1 - \frac{\lambda_p}{\lambda_q} = 0.8548$. This is very close to the value we got from Equation 4.12, 0.8551, meaning that there are more than 85% peers joining multiple torrents.

Having characterized the torrent request pattern of peers, finally we consider the distribution of the seeding time and the sleeping time of peers. According to our fluid model, $\frac{1}{\gamma}$ represents the average seeding time. Figure 13(a) and 13(b) show the probability distribution functions of the peer seeding time and the peer sleeping time in the system. Note that the y-axis is in log scale. Both the peer seeding time and sleeping time roughly follow the exponential distribution with probability density function $f_{sd}(t) = \frac{1}{\tau_{sd}}e^{-\frac{t}{\tau_{sd}}}$, and $f_{sl}(t) = \frac{1}{\tau_{sl}}e^{-\frac{t}{\tau_{sl}}}$, respectively. Based on the trace analysis, we estimate $\tau_{sd} = \frac{1}{\gamma} = 8.42$ hours, and $\tau_{sl} = 58.32$ hours.

4.2 Characterizing the Inter-Torrent Relation

In this part we study how different torrents are connected through peers that download multiple files, based on our previously verified assumptions.

For simplification, we consider a homogeneous multi-torrent environment where all torrents and peers have the same λ_0, τ, μ, c, η, γ, and average sleeping time. We denote each torrent in the system as torrent i with birth time t_i ($1 \le i < \infty$). For any two torrents that are born successively, torrent j first born and torrent i born next, we have $i = j+1$ and $t_i > t_j$.

Assume the probability that a peer selects torrent i at time t_0 as its k-th torrent is $P_i^k(t_0)$, $P_i^k(t_0) = 0$ when $t_i > t_0$. We also denote $P_i^1(t_0)$ as $P_i(t_0)$ for simplicity. Without loss of generality, we assume that the most recently born torrent by time t_0 is torrent 1, and $P_i(t_0)$ satisfies

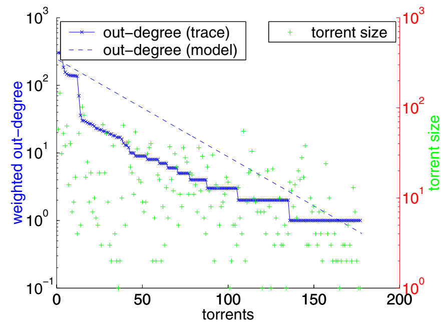
$$P_i(t_0) = \frac{e^{-\frac{t_0-t_i}{\tau}}}{\sum_{j=1}^{\infty} e^{-\frac{t_0-t_j}{\tau}}}, \tag{4.15}$$

where $t_j = t_0 - \frac{j}{\lambda_t}$, $1 \le j < \infty$. Thus, we have

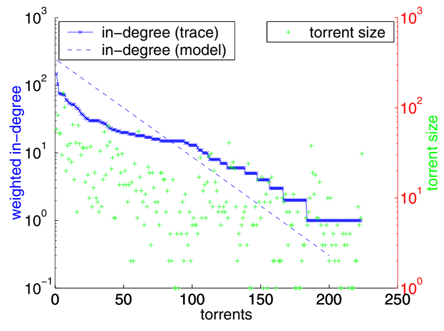
$$\begin{aligned} P_i(t_0) = \frac{e^{-\frac{i}{\lambda_t\tau}}}{\sum_{j=1}^{\infty} e^{-\frac{j}{\lambda_t\tau}}} &= (e^{\frac{1}{\lambda_t\tau}} - 1)e^{-\frac{i}{\lambda_t\tau}} \\ &= (e^{\frac{1}{\lambda_t\tau}} - 1)e^{-\frac{t_0-t_i}{\tau}}. \end{aligned} \tag{4.16}$$

For a peer that requests its k-th torrent file, the peer does not select the torrents that it has requested. Assuming

$$P_i^k(t_0) = \alpha_k P_i(t_0), \tag{4.17}$$



(a) Weighted out-degree



(b) Weighted in-degree

Figure 14: The inter-torrent relation (y-axis is in log scale)

the peer arrival rate of a torrent can be expressed as

$$\begin{aligned}\lambda(t) &= \alpha\lambda_q P_i(t_0)\\ &= \frac{\alpha}{1-p}\lambda_p(e^{\frac{1}{\lambda_t\tau}}-1)e^{-\frac{t-t_i}{\tau}},\end{aligned} \tag{4.18}$$

where $\alpha = \sum_{k=1}^{\infty}\alpha_k p^{k-1}(1-p)$. When $\lambda_t \gg r$, we have $\alpha_k \approx 1$ and $\alpha \approx 1$. Comparing Equation 3.1 with 4.18, we have $\lambda_0 = \frac{\alpha}{1-p}\lambda_p(e^{\frac{1}{\lambda_t\tau}}-1)$.

Considering that a peer in a torrent may have downloaded files from other torrents, we can model the relationship among different torrents in the P2P system as a *directed graph*. Each node in the graph represents a torrent. A directed edge from torrent i to torrent j denotes that some peers in torrent i have downloaded the file from torrent j, and thus have the potential to provide service to peers in torrent j, even though they are not in torrent j currently. The weight of the directed edge $W_{i,j}$ represents the number of such peers. For simplicity, we define $W_{i,i} = 0$.

The graph changes dynamically over time. Now let us consider the graph at time t_0. During time $[t, t+dt]$, $t_j \le t < t_0$, there are $\lambda(t)dt$ peers who joined torrent j. Let $k(t) = \lfloor r(t_0-t)\rfloor$. During time $[t, t_0]$, these peers can download up to $k(t)-1$ torrents completely in addition to torrent j and may request (or be requesting) the next torrent at time t_0. If torrent i is not requested before the last requests during time $[t, t_0]$, the probability of such events is

$$Q_i(t) = p \times \prod_{l=1}^{k(t)-1} p \times (1-\alpha_l P_i(t+\frac{l}{r})). \tag{4.19}$$

When $i \neq j$, we have

$$W_{i,j} = \int_{t_j}^{t_0}\lambda(t)dt \times Q_i(t) \times \alpha_{k(t)}P_i(t+\frac{k(t)}{r}). \tag{4.20}$$

Therefore, the weighted out-degree of torrent i represents the total potential capability its peers can provide to peers in other torrents, denoted as SP_i, where

$$SP_i = \sum_{j=1}^{\infty} W_{i,j}. \tag{4.21}$$

Correspondingly, the weighted in-degree of torrent i represents the total potentials its peers can get from peers in other torrents, denoted as SG_j, where

$$SG_j = \sum_{i=1}^{\infty} W_{i,j}. \tag{4.22}$$

Figure 14(a) and 14(b) show the weighted out-degree and weighted in-degree at a time instant based on trace analysis and our probability model, respectively. In the figures, each point in the x-axis denotes a torrent, sorted in non-ascending order of weighted out-degree or weighted in-degree. The right y-axis in the figures denotes *torrent size*, the number of peers in the torrent at this time instant. In general, torrents with more peers tend to have large out-degree and in-degree. The weighted out-degree and in-degree distribution according to our trace analysis follows power law rules roughly. It deviates from our model somewhat because of the heterogeneity of torrents in the real system.

In the multi-torrent environment, old peers that had downloaded the file from a torrent may come back for other torrent files, and the lifespan of this torrent can be extended if these old peers are willing to provide service. Assume the request arrival rate of this torrent is $\lambda(t)$ and $\lambda(t) = 0$ when $t < 0$. If we consider both new requesting peers and old returning peers, the peer arrival rate of the torrent is

$$\begin{aligned}\lambda'(t) &= \textstyle\sum_{l=0}^{k(t)} p^l\lambda(t-\frac{l}{r}) = \sum_{l=0}^{k(t)} p^l\lambda_0 e^{-\frac{t-\frac{l}{r}}{\tau}}\\ &= \lambda_0 e^{-\frac{t}{\tau}}\frac{q^{k(t)+1}-1}{q-1},\end{aligned} \tag{4.23}$$

where $k(t) = \lfloor rt\rfloor$ and $q = pe^{\frac{1}{r\tau}}$ ($q > 1$ based on our trace analysis).

When $\lambda'(t) < \gamma$, the torrent is truly dead. The lifespan of a torrent without inter-torrent collaboration is $T_{life} = \tau \log(\frac{\lambda_0}{\gamma})$. Denoting the lifespan of the torrent with inter-torrent collaboration as T'_{life}, then $\lambda'(T'_{life}) = \gamma$, we have

$$\begin{aligned}\log \gamma &= \log \lambda_0 - \frac{T'_{life}}{\tau} + \log(q^{k(T'_{life})+1} - 1) - \log(q-1) \\ &\approx \log \lambda_0 - \frac{T'_{life}}{\tau} + (k(T'_{life}) + 1) \log q - \log(q-1) \\ &= \log \lambda_0 - \frac{T'_{life}}{\tau} + k(T'_{life}) \log q + \log \frac{q}{q-1}.\end{aligned}$$

It leads to $\log(\frac{\lambda_0}{\gamma} \frac{q}{q-1}) \approx (\frac{1}{\tau} - r \log q) T'_{life}$. Thus

$$\begin{aligned}T'_{life} &\approx \frac{\tau \log(\frac{\lambda_0}{\gamma} \frac{q}{q-1})}{1 - \tau r \log q} = \frac{\tau \log(\frac{\lambda_0}{\gamma} \frac{q}{q-1})}{\tau r \log \frac{1}{p}} \\ &> \frac{T_{life}}{\tau r \log \frac{1}{p}} = \beta T_{life}.\end{aligned} \tag{4.24}$$

According to the trace analysis and our modeling, $\beta = \frac{1}{\tau r \log \frac{1}{p}} \approx 6$. So we have

$$R'_{fail} = e^{-\frac{T'_{life}}{\tau}} < R^{\beta}_{fail} \approx R^{6}_{fail}. \tag{4.25}$$

Comparing Equation 4.25 with 3.6, we can see that inter-torrent collaboration is much more effective than decreasing the seed leaving rate γ for reducing downloading failure ratio. Decreasing seeds leaving rate has polynomial effect, while inter-torrent collaboration has exponential effect. For example, if the current downloading failure rate is 0.1, and seeds can be stimulated to stay 10 times longer (i.e., γ will decrease 10 times), then the downloading failure rate will decrease 10 times to 0.01. However, by inter-torrent collaboration, the downloading failure ratio can be as low as $0.1^6 = 10^{-6}$. The reason is that extending seed staying time only increases the service time for peers that arrive close to the seed generation time. With the passage of time, the peer arrival rate decreases exponentially, and finally the seed serving time will not be long enough for newly arriving peers. On the other hand, by exploiting inter-torrent collaboration, peers that have downloaded the file may return multiple times during a much longer period, and the downloading failure ratio can be significantly reduced to near zero.

5 A Discussion of Multi-Torrent Collaboration Systems

In this section, we discuss the principle of a system design for multi-torrent collaboration. A more detailed discussion can be found in [13]. The system design and implementation are ongoing.

5.1 Tracker Site Overlay

In BitTorrent systems, peers in different torrents cannot collaborate because they cannot find and communicate with each other. The inter-torrent relation graph presented in Section 4.2 motivates us to organize the tracker sites of different torrents into an overlay network to help the peers sharing different files find each other and coordinate the collaboration among these peers. In such an overlay network, each tracker site maintains a *Neighbor-Out Table* and a *Neighbor-In Table* to record the relationship with its neighboring torrents. The *Neighbor-Out Table* records the torrents that its peers can provide service to. The *Neighbor-In Table* records the torrents whose peers can provide service to this torrent. When a peer q joins a new torrent A, it uploads to its tracker site the information about from which torrents it had downloaded files previously. Then A's tracker site forwards this information to the tracker sites of those torrents where q had downloaded files from. By doing so, the torrents that are created independently by different content providers are connected together to form a *tracker site overlay*, as shown in Figure 15. Tracker site overlay also provides a built-in mechanism to search content among multiple torrents. Currently, BitTorrent users have to rely on Web-based search engines to look for the content they want to download.

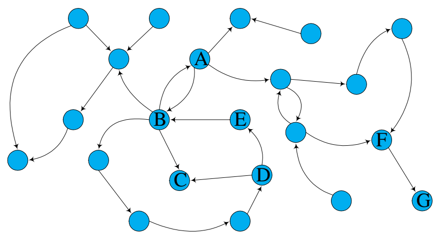


Figure 15: Tracker site overlay

5.2 Exchange Based Incentive for Multi-torrent Collaboration

BitTorrent assumes each peer is selfish, and exchanges file chunks with those peers that provide it the best service. The incentive mechanism in BitTorrent systems is instant, because each peer must get corresponding benefit at once for the service it provides. For multi-torrent collaboration, an exchange based mechanism can be applied for *instant collaboration* through the tracker site overlay, which still follows the “tit-for-tat” idea.

First, peers in adjacent torrents in the overlay can exchange file chunks directly, such as torrent A, B in Figure 15. Second, if there exists a cycle among several torrents, then peers in adjacent torrents can exchange file chunks through the coordination of the tracker site overlay, such as torrent B, C, D, E in Figure 15. More specifically, when a peer q wants to get service from peers in other torrents, it sends a request to its tracker site with its list of downloaded files. Then the tracker forwards its request to the trackers in its *Neighbor-In Table*. These tracker sites then search their tables to find qualified peers, with whom this peer can exchange file chunks to get service.

When a peer q wants to get service from peers in other torrents and it has no service to exchange, it may join these torrents temporarily and download some chunks of the files, even if it does not want these files itself. Through the coordination of corresponding tracker sites, the peer can provide uploading service for these chunks only, and attribute its service contribution to the peers it wants to get service from, so that these peers can get benefit from the peers that q serves and offer q the service it needs. Since a file chunk can be served to multiple peers in the system, this method is very effective and the overhead is trivial. Research [6, 9] presents similar idea of using file exchange as an incentive for P2P content sharing. Different from these studies, our system aims to share bandwidth as well as content across multiple P2P systems.

6 Conclusion

BitTorrent-like systems have become increasingly popular for object distribution and file sharing, and have contributed to a large amount of traffic on the Internet. In this paper, we have performed extensive trace analysis and modeling to study the behaviors of such systems. We found that the existing BitTorrent system provides poor service availability, fluctuating downloading performance, and unfair services to peers. Our model has revealed that these problems are due to the exponentially decreasing peer arrival rate and provides strong motivation for inter-torrent collaborations instead of simply giving seeds incentives to stay longer. We also discuss the design of a new system where the tracker sites of different torrents are organized into an overlay to facilitate inter-torrent collaboration with the help of an exchange based incentive mechanism.

7 Acknowledgments

We would like to thank our shepherd, Keith W. Ross, and the anonymous reviewers for their constructive comments. We are grateful to Oliver Spatscheck for many insightful discussions on this topic. Mikel Lzal helped us get access to their BitTorrent traces and William L. Bynum provided valuable comments on an early draft of the paper. This work is partially supported by the National Science Foundation under grants CNS-0098055, CNS-0405909, and CNS-0509054/0509061.

References

[1] http://www.gnutelliums.com/.

[2] http://www.kazaa.com/.

[3] http://www.edonkey2000.com/.

[4] Hack kazaa participation level - the easy answer. http://www.davesplanet.net/kazaa/.

[5] ADAR, E., AND B.HUBERMAN. Free riding on gnutella. Tech. rep., Xerox PARC, August 2000.

[6] ANAGNOSTAKIS, K. G., AND GREENWALD, M. B. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *Proc. of IEEE ICDCS* (March 2004).

[7] BELLISSIMO, A., LEVINE, B. N., AND SHENOY, P. Exploring the use of BitTorrent as the basis for a large trace repository. Tech. Rep. 04-41, University of Massachusetts Amherst, June 2004.

[8] COHEN, B. Incentives build robustness in BitTorrent. In *Proc. of Workshop on Economics of Peer-to-Peer Systems* (May 2003).

[9] COX, L. P., AND NOBLE, B. D. Samsara - honor among thieves in P2P storage. In *Proc. of ACM SOSP* (October 2003).

[10] CRANOR, C., JOHNSON, T., AND SPATSCHECK, O. Gigascope: a stream database for network applications. In *Proc. of ACM SIGMOD* (June 2003).

[11] GE, Z., FIGUEIREDO, D. R., JAISWAL, S., KUROSE, J., AND TOWSLEY, D. Modeling peer-peer file sharing systems. In *Proc. of IEEE INFOCOM* (March 2003).

[12] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP* (October 2003).

[13] GUO, L., CHEN, S., XIAO, Z., TAN, E., DING, X., AND ZHANG, X. Measurements, analysis, and modeling of BitTorrent systems. Tech. Rep. WM-CS-2005-08, College of William and Mary, July 2005.

[14] IZAL, M., URVOY-KELLER, G., BIERSACK, E., FELBER, P., HAMRA, A. A., AND GARC'ES-ERICE, L. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Proc. of the 5th Annual Passive & Active Measurement Workshop* (April 2004).

[15] MASSOULIE, L., AND VOJNOVIC, M. Coupon replication systems. In *Proc. of ACM SIGMETRICS* (June 2005).

[16] PARKER, A. The true picture of peer-to-peer file sharing. http://www.cachelogic.com, 2004.

[17] POUWELSE, J., GARBACKI, P., EPEMA, D., AND SIPS, H. The BitTorrent P2P file-sharing system: Measurements and analysis. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems* (February 2005).

[18] QIU, D., AND SRIKANT, R. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. of ACM SIGCOMM* (August 2004).

[19] SAROIU, S., GUMMADI, K., DUNN, R., GRIBBLE, S., AND LEVY, H. An analysis of Internet content delivery systems. In *Proc. of USENIX OSDI* (December 2002).

[20] SAROIU, S., GUMMADI, K., AND GRIBBLE, S. A measurement study of peer-to-peer file sharing systems. In *Proc. of ACM/SPIE MMCN* (January 2002).

[21] SHERWOOD, R., BRAUD, R., AND BHATTACHARJEE, B. Slurpie: A cooperative bulk data transfer protocol. In *Proc. of IEEE INFOCOM* (March 2004).

[22] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. Efficient content location using interest-based locality in peer-to-peer systems. In *Proc. of IEEE INFOCOM* (March 2003).

[23] YANG, X., AND D. VECIANA, G. Service capacity of peer to peer networks. In *Proc. of IEEE INFOCOM* (March 2004).

Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems

Daniel Stutzbach, Reza Rejaie
University of Oregon
{*agthorr,reza*}*@cs.uoregon.edu*

Subhabrata Sen
AT&T Labs—Research
sen@research.att.com

Abstract

During recent years, peer-to-peer (P2P) file-sharing systems have evolved in many ways to accommodate growing numbers of participating peers. In particular, new features have changed the properties of the unstructured overlay topology formed by these peers. Despite their importance, little is known about the characteristics of these topologies and their dynamics in modern file-sharing applications.

This paper presents a detailed characterization of P2P overlay topologies and their dynamics, focusing on the modern Gnutella network. Using our fast and accurate P2P crawler, we capture a complete snapshot of the Gnutella network with more than one million peers in just a few minutes. Leveraging more than 18,000 recent overlay snapshots, we characterize the graph-related properties of individual overlay snapshots and overlay dynamics across hundreds of back-to-back snapshots. We show how inaccuracy in snapshots can lead to erroneous conclusions—such as a power-law degree distribution. Our results reveal that while the Gnutella network has dramatically grown and changed in many ways, it still exhibits the clustering and short path lengths of a small world network. Furthermore, its overlay topology is highly resilient to random peer departure and even systematic attacks. More interestingly, overlay dynamics lead to an "onion-like" biased connectivity among peers where each peer is more likely connected to peers with higher uptime. Therefore, long-lived peers form a stable core that ensures reachability among peers despite overlay dynamics.

1 Introduction

The Internet has witnessed a rapid growth in the popularity of various Peer-to-Peer (P2P) applications during recent years. In particular, today's P2P file-sharing applications (*e.g.*, FastTrack, eDonkey, Gnutella) are extremely popular with millions of simultaneous clients and contribute a significant portion of the total Internet traffic [1, 13, 14]. These applications have changed in many ways to accommodate growing numbers of participating peers. In these applications, participating peers form an overlay which provides connectivity among the peers to search for desired files. Typically, these overlays are *unstructured* where peers select neighbors through a predominantly random process, contrasting with *structured* overlays, *i.e.*, distributed hash tables such as Chord [29] and CAN [22]. Most modern file-sharing networks use a *two-tier* topology where a subset of peers, called *ultrapeers*, form an unstructured mesh while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple ultrapeers. More importantly, the overlay topology is continuously reshaped by both user-driven dynamics of peer participation as well as protocol-driven dynamics of neighbor selection. In a nutshell, as participating peers join and leave, they collectively, in a decentralized fashion, form an unstructured and dynamically changing overlay topology.

The design and simulation-based evaluation of new search and replication techniques has received much attention in recent years. These studies often make certain assumptions about topological characteristics of P2P networks (*e.g.*, power-law degree distribution) and usually ignore the dynamic aspects of overlay topologies. However, little is known about the topological characteristics of popular P2P file sharing applications, particularly about overlay dynamics. An important factor to note is that properties of unstructured overlay topologies cannot be easily derived from the neighbor selection mechanisms due to implementation heterogeneity and dynamic peer participation. Without a solid understanding of topological characteristics in file-sharing applications, the actual performance of the proposed search and replication techniques in practice is unknown, and cannot be meaningfully simulated.

Accurately characterizing the overlay topology of a large scale P2P network is challenging [33]. A common approach is to examine properties of snapshots of the overlay captured by a topology crawler. However, capturing accurate snapshots is inherently difficult for two reasons: *(i)* the dynamic nature of overlay topologies, and *(ii)* a non-negligible fraction of discovered peers in each snapshot are not directly reachable by the crawler. Furthermore, the accuracy of captured snapshots is difficult to verify due to the lack of any accurate reference snapshot.

Previous studies that captured P2P overlay topologies with a crawler either deployed slow crawlers, which inevitably lead to significantly distorted snapshots of the overlay [23], or partially crawled the overlay [24, 18] which is likely to capture biased (and non-representative) snapshots. These studies have not examined the accuracy of their captured snapshots and only conducted limited analysis of the overlay topology. More importantly, these few studies (except [18]) are outdated (more than three years old) since P2P filesharing applications have significantly increased in size and incorporated several new topological features over the past few years. An interesting recent study [18] presented a high level characterization of the two-tier Kazaa overlay topology. However, the study does not contain detailed graph-related properties of the overlay. Finally, to our knowledge, the dynamics of unstructured P2P overlay topologies have not been studied in detail in any prior work.

We have recently developed a set of measurement techniques and incorporated them into a parallel P2P crawler, called *Cruiser* [30]. Cruiser can accurately capture a complete snapshot of the Gnutella network with more than one million peers in just a few minutes. Its speed is several orders of magnitude faster than any previously reported P2P crawler and thus its captured snapshots are significantly more accurate. Capturing snapshots rapidly also allows us to examine the dynamics of the overlay over a much shorter time scale, which was not feasible in previous studies. *This paper presents detailed characterizations of both graph-related properties as well as the dynamics of unstructured overlay topologies based on recent large-scale and accurate measurements of the Gnutella network.*

1.1 Contributions

Using Cruiser, we have captured more than 18,000 snapshots of the Gnutella network during the past year. We use these snapshots to characterize the Gnutella topology at two levels:

- *Graph-related Properties of Individual Snapshots*: We treat individual snapshots of the overlay as graphs and apply different forms of graph analysis to examine their properties[1].
- *Dynamics of the Overlay*: We present new methodologies to examine the dynamics of the overlay and its evolution over different timescales.

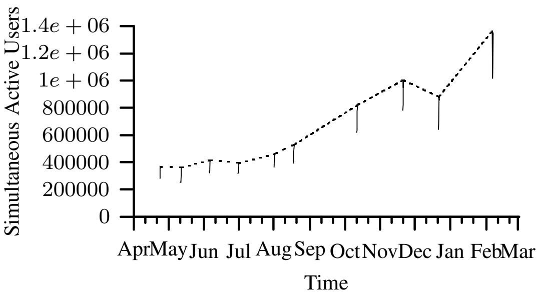


Figure 1: Change in network size over months. Vertical bars show variation within a single day.

We investigate the underlying causes of the observed properties and dynamics of the overlay topology. To the extent possible, we conduct our analysis in a generic (*i.e.*, Gnutella-independent) fashion to ensure applicability to other P2P systems. Our main findings can be summarized as follows:

- In contrast to earlier studies [7, 23, 20], we find that node degree does not exhibit a power-law distribution. We show how power-law degree distributions can result from measurement artifacts.
- While the Gnutella network has dramatically grown and changed in many ways, it still exhibits the clustering and the short path lengths of a small world network. Furthermore, its overlay topology is highly resilient to random peer departure and even systematic removal of high-degree peers.
- Long-lived ultrapeers form a stable and densely connected *core overlay*, providing stable and efficient connectivity among participating peers despite the high degree of dynamics in peer participation.
- The longer a peer remains in the overlay, the more it becomes clustered with other long-lived peers with similar uptime[2]. In other words, connectivity within the core overlay exhibits an "onion-like" bias where most long-lived peers form a well-connected core, and a group of peers with shorter uptime form a layer with a relatively biased connectivity to each other and to peers with higher uptime (*i.e.*, internal layers).

1.2 Why Examine Gnutella?

eDonkey, FastTrack, and Gnutella are the three most popular P2P file-sharing applications today, according to Slyck.com [1], a website which tracks the number of users for different P2P applications. We elected to first focus on the Gnutella network due to a number of considerations.

First, a variety of evidence indicates that the Gnutella network has a large and growing population of active users and generates considerable traffic volume. Figure 1 depicts the average size of the Gnutella network over an eleven month period ending February 2005, indicating that network size has more than tripled (from 350K to 1.3 million peers) during our measurement period. We also observed time-of-day effects in the size of captured snapshots, which is a good indication of active user participations in the Gnutella network. Also, examination of Internet2 measurement logs[3] reveal that the estimated Gnutella traffic measured on that network is considerable and growing. For example, for the 6 week period $10/11/04 - 11/21/04$, the Gnutella traffic on Internet2 was estimated to be 79.69 terabytes, up from 21.52 terabytes for a 6 week period ($02/02/04 - 03/14/04$) earlier that year.

Second, Gnutella, which was the first decentralized P2P system, has evolved significantly since its inception in 2000. While it is among the most studied P2P networks in the literature, prior studies are at least 2–3 years old, and mostly considered the earlier flat-network incarnation. A detailed measurement study of the modern two-tier Gnutella network is therefore timely and allows us to compare and contrast the behavior today from the earlier measurement studies, and to gain insights into the behavior and impact of the two-tier, unstructured overlay topologies which have been adopted by most modern P2P systems.

Third, our choice was also influenced by the fact that Gnutella is the most popular P2P file-sharing network with an open and well-documented protocol specification. This eliminates (or at least significantly reduces) any incompatibility error in our measurement that could potentially occur in other proprietary P2P applications that have been reverse-engineered, such as FastTrack/Kazaa and eDonkey.

The rest of this paper is organized as follows: Section 2 provides a description of the modern Gnutella P2P overlay network and describes the fundamental challenges in capturing accurate snapshots. We present a brief overview of our crawler in Section 3. Section 4 presents a detailed characterization of graph-related properties of individual snapshots as well as the implications of our findings. In Section 5, we examine overlay dynamics, their underlying causes, and their implications on design and evaluation of P2P applications. Section 6 presents an overview of related work and Section 7 concludes the paper.

2 Background

To accurately characterize P2P overlay topologies, we need to capture *complete* and *accurate* snapshots. By "snapshot", we refer to a graph that presents all participating peers (as nodes) and the connections between them (as edges) at a single instance in time. The most reliable, and thus common, approach to capture a snapshot is to crawl the overlay. Given information about a handful of initial peers, the crawler progressively contacts participating peers and collects information about their neighbors. In practice, capturing accurate snapshots is challenging for two reasons:

(i) The Dynamic Nature of Overlays: Crawlers are not instantaneous and require time to capture a complete snapshot. Because of the dynamic nature of peer participation and neighbor selection, the longer a crawl takes, the more changes occur in participating peers and their connections, and the more *distorted* the captured snapshot becomes. More specifically, any connection that is established or closed during a crawl (*i.e.*, *changing connections*) is likely to be reported only by one end of the connection. We note that there is no reliable way to accurately resolve the status of changing peers or changing connections. In a nutshell, any captured snapshot by a crawler will be distorted, where the degree of distortion is a function of the crawl duration relative to the rate of change in the overlay.

(ii) Unreachable Peers: A significant portion of discovered peers in each snapshot are not directly reachable since they have departed, reside behind a firewall, or are overloaded [30]. Therefore, information about the edges of the overlay that are connected between these unreachable peers will be missing from the captured snapshots.

We argue that sampling a snapshot of unstructured networks through partial crawls [24] or passive monitoring [25] is not a reliable technique for an initial characterization of the overlay topology for the following reasons: *(i)* in the absence of adequate knowledge about the properties and dynamics of the overlay topology, it is difficult to collect unbiased samples. For example, partial crawling of the network can easily result in a snapshot that is biased towards peers with higher degree; *(ii)* some graph-level characteristics of the overlay topology, such as the mean shortest path between peers (which we discuss in Subsection 4.2) cannot be accurately derived from partial snapshots. Because of these reasons, we attempt to capture snapshots as complete as possible and use them for our characterizations.

To describe our measurement methodology for addressing the above challenges, we provide a brief description of modern Gnutella as an example of a two-tier P2P file-sharing application.

2.1 Modern Gnutella

In the original Gnutella protocol, participating peers form a flat unstructured overlay and use TTL-scoped flooding of search queries to other peers. This approach has limited scalability. To improve the scalability of the Gnutella protocol, most modern Gnutella clients adopt a new overlay structure along with a new query distribution mechanism as follows:

(i) Two-tier Overlay: A new generation of popular file-sharing applications have adopted a *two-tier* overlay archi-

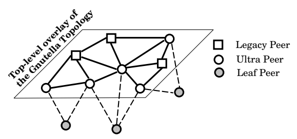


Figure 2: Two-tier Topology of Modern Gnutella

tecture to improve their scalability: a subset of peers, called *ultrapeers*, form a *top-level* overlay while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple ultrapeers (Figure 2). FastTrack (or Kazaa), Gnutella, and eDonkey all use some variation of this model. Those peers that do not implement the ultrapeer feature, called *legacy* peers, can only reside in the top-level overlay and do not accept any leaves. When a leaf connects to an ultrapeer, it uploads a set of hashes of its filename keywords to that ultrapeer. This allows the ultrapeer to only forward messages to the leaves who might have matching files. This approach reduces the number of forwarded messages towards leaf peers which in turn increases the scalability of the network by a constant factor. Leaf peers never forward messages.
(ii) Dynamic Query: The Gnutella developer community has adopted a new scheme for query distribution called *Dynamic Querying* [9]. The goal in this scheme is to only gather enough results to satisfy the user (typically 50 to 200 results). Rather than forwarding a query to all neighbors, ultrapeers manage the queries for their leaves. Toward this end, an ultrapeer begins by forwarding a query to a subset of top-level connections using a low TTL. From that point on, the query is flooded outward until the TTL expires. The ultrapeer then waits for the results, and uses the ratio between the number of results and the estimated number of visited peers to determine how rare matches are. If matches are rare (*i.e.*, there are few or no responses), the query is sent through more connections with a relatively high TTL. If matches are more common but not sufficient, the query is sent down a few more connections with a low TTL. This process is repeated until the desired number of results are collected or the ultrapeer gives up. Each ultrapeer estimates the number of visited ultrapeers through each neighbor based on the following formula: $\sum_{i=0}^{\mathrm{TTL}-1}(d-1)^i$. This formula assumes that all peers have the same node degree, d. When Dynamic Querying was introduced, the number of neighbors each ultrapeer attempts to maintain was increased to allow more fine-grained control with Dynamic Querying by giving ultrapeers more neighbors to choose from.

3 Capturing Accurate Snapshots

In this section, we present an overview of our data collection and post-processing steps.

Cruiser: We have developed a set of measurement techniques into a parallel Gnutella crawler, called *Cruiser* [30]. While the basic crawling strategy by Cruiser is similar to other crawlers, it improves the accuracy of captured snapshots by significantly increasing the crawling speed (*i.e.*, reducing crawl duration) primarily by using the following techniques: First, Cruiser employs a master-slave architecture in order to achieve a high degree of concurrency and to effectively utilize available resources on multiple PCs. Using a master-slave architecture also allows us to deploy Cruiser in a distributed fashion if Cruiser's access link becomes a bottleneck. The master process coordinates multiple slave processes that crawl disjoint portions of the network in parallel. Each slave crawler opens hundreds of parallel connections, contributing a speed-up of nearly three orders of magnitude.

Second, Cruiser leverages the two-tier structure of the modern Gnutella network by only crawling the top-level peers (*i.e.*, ultrapeers and legacy peers). Since each leaf must be connected to an ultrapeer, this approach enables us to capture all the nodes and links of the overlay by contacting a relatively small fraction of all peers. Overall, this strategy leads to around an 85% reduction in the duration of a crawl without any loss of information.

These techniques collectively result in a significant increase in crawling speed. *Cruiser can capture the Gnutella network with one million peers in around 7 minutes using six off-the-shelf 1 GHz GNU/Linux boxes in our lab. Cruiser's crawling speed is about 140K peers/minute (by directly contacting 22K peers/minute), This is orders of magnitude faster than previously reported crawlers (*i.e.*, 2 hours for 30K peers (250/minute) in [23], and 2 minutes for 5K peer (2.5K/minute) in [24]).* It is worth clarifying that while our crawling strategy is aggressive and our crawler requires considerable local resources, its behavior is not intrusive since each top-level peer is contacted only once per crawl.

Post-Processing: Once information is collected from all reachable peers, we perform some post-processing to remove any obvious inconsistencies that might have been introduced due to changes in the topology during the crawling period. Specifically, we include edges even if they are only reported by one peer, and treat a peer as an ultrapeer if it neighbors with another ultrapeer or has any leaves. Due to the inconsistencies, we might over-count edges by about 1% and ultrapeers by about 0.5%.

Unreachable Peers: We have carefully examined the effect of unreachable peers on the accuracy of captured snapshots [33]. Previous studies assumed that these unreachable peers departed the network or are legacy peers that reside behind a firewall (or NAT), and simply excluded this large group of unreachable peers from their snapshot. It is important to determine what portion of unreachable peers are departed or NATed because each group introduces a different

Crawl Date	Total Nodes	Leaves	Top-level	Unreachable	Top-Level Edges
09/27/04	725,120	614,912	110,208	35,796	1,212,772
10/11/04	779,535	662,568	116,967	41,192	1,244,219
10/18/04	806,948	686,719	120,229	36,035	1,331,745
02/02/05	1,031,471	873,130	158,345	39,283	1,964,121

Table 1: Sample Crawl Statistics

error on the snapshot. However, there is no reliable test to distinguish between departed and firewalled peers because firewalls can time out or refuse connections depending on their configuration.

In summary, our investigation revealed that in each crawl, 30%–38% of discovered peers are unreachable. In this group, the breakdown is as follows: 2%–3% are departed peers, 15%–24% are firewalled, and the remaining unreachable peers (3%–21%) are either also firewalled or overwhelmed ultrapeers. However, since Cruiser only needs to contact *either* end of an edge, it is able to discover at least 85%–91% of edges. Since firewalled peers cannot directly connect together (*i.e.*, cannot be located at both ends of a missing edge) and they constitute more than half of the unreachable peers, the actual portion of missing edges is considerably smaller.

Quantifying Snapshot Accuracy: We rigorously examined the effect of crawling speed and duration on two dimensions of snapshot accuracy: completeness and distortion. Our evaluations [30] revealed that *(i)* Cruiser captures nearly all ultrapeers and the pair-wise connections between them and the majority of connections to leaves; *(ii)* Both node distortion and edge distortion in captured snapshots increases linearly with the crawl duration; and *(iii)* snapshots captured by Cruiser have little distortion. For example, we found that two back-to-back snapshots differed only 4% in their peer populations.

Data Set: We have captured more than 18,000 snapshots of the Gnutella network during the past eleven months (Apr. 2004–Feb. 2005) with Cruiser. In particular, we collected back-to-back snapshots for several one-week intervals as well as randomly distributed snapshots during various times of the day to ensure that captured snapshots are representative. In Section 4, we use four of these snapshots to illustrate graph properties of the overlay topology. In Section 5, we use sets of hundreds of back-to-back snapshots to examine how the overlay topology evolves with time.

4 Overlay Graph Properties

The two-tier overlay topology in modern Gnutella (as well as other unstructured P2P networks) consists of ultrapeers that form a "spaghetti-like" top-level overlay and a large group of leaf peers that are connected to the top-level through multiple ultrapeers. We treat individual snapshots of the overlay as graphs and apply different forms of graph analysis to examine their properties. We pay special attention to the top-level overlay since it is the core component of the topology. Throughout our analysis, we compare our findings with similar results reported in previous studies. However, it is important to note that we are unable to determine whether the reported differences (or similarities) are due to changes in the Gnutella network or due to inaccuracy in the captured snapshots of previous studies.

Implementation:	LimeWire	BearShare	Other
Percentage:	74%–77%	19%–20%	4%–6%

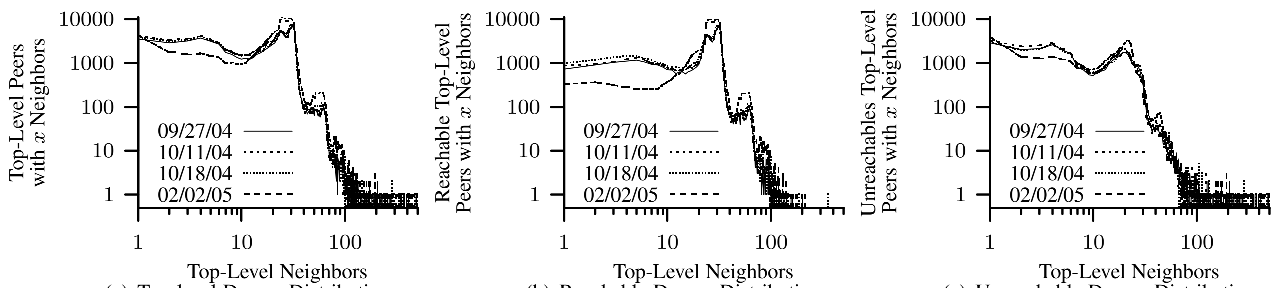
Table 2: Distribution of Implementation

Table 1 presents summary information of four sample snapshots after post-processing. The results in this section are primarily from the snapshots in Table 1. However, we have examined many other snapshots and observed similar trends and behaviors. Therefore, we believe the presented results are representative. Presenting different angles of the same subset of snapshots allows us to conduct cross comparisons and also relate various findings.

In this section, we explore the node degree distribution in Subsection 4.1, the reachability and pairwise distance properties of the overlay in Subsection 4.2, small world characteristics in Subsection 4.3, and the resilience of the overlay in Subsection 4.4.

Implementation Heterogeneity: The open nature of the Gnutella protocol has led to several known (and possibly many unknown) implementations. It is important to determine the distribution of different implementations (and configurations) among participating peers since their design choices directly affect the overall properties of the overlay topology. This will help us explain some of the observed properties of the overlay. Table 2 presents the distribution of different implementations across discovered ultrapeers. This table shows that a clear majority of contacted ultrapeers use the LimeWire implementation. We also discovered that a majority of LimeWire ultrapeers (around 94%) use the most recent version of the software available at the time of the crawl. These results reveal that while heterogeneity exists, nearly all Gnutella users run LimeWire or BearShare.

We are particularly interested in the number of connec-



(a) Top-level Degree Distribution (b) Reachable Degree Distribution (c) Unreachable Degree Distribution

Figure 3: Different angles of the top-level degree distribution in Gnutella topology

tions that are used by each implementation since this design choice directly affects the degree distribution of the overall topology. This information can be obtained from available LimeWire source code. However, not all implementations are open, and users can always change the source code of open implementations. Thus, we need to collect this information from running ultrapeers in action.

Our measurements reveal that LimeWire's and BearShare's ultrapeer implementations prefer to serve 30 and 45 leaves, respectively, whereas both try to maintain around 30 neighbors in the top-level overlay.

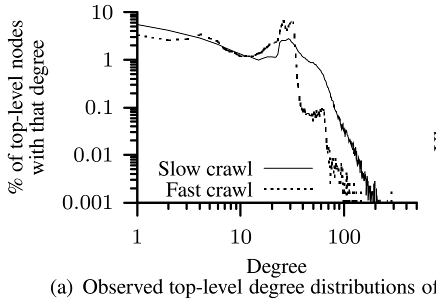
4.1 Node Degree Distributions

The introduction of the two-tier architecture in the overlay topology along with the distinction between ultrapeers and leaf peers in the modern Gnutella protocol demands a close examination of the different degree distributions among different group of peers.

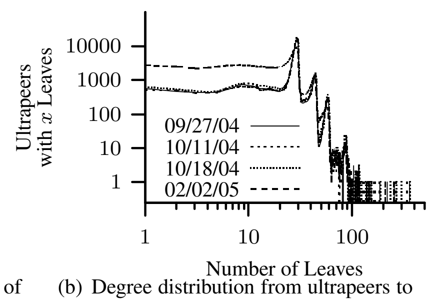
Node Degree in the Top-Level Overlay: Previous studies reported that the distribution of node degree in the Gnutella network exhibited a power-law distribution [23, 2, 7] and later changed to a two-segment power-law distribution [20, 23]. To verify this property for the modern Gnutella network, Figure 3(a) depicts the distribution of node degree among all peers (both unreachable and reachable) in the top-level overlay for the four sample snapshots presented in Table 1. This distribution has a spike around 30 and does not follow a power-law[4]. A key question is *to what extent this difference in degree distribution is due to the change in the overlay structure versus error in captured snapshots by earlier studies.* To examine this question, we captured a distorted snapshot by a slow crawler[5] which is similar to the 50-connection crawler used in an earlier study [23]. Figure 4(a) depicts the degree distribution based on this distorted snapshot, which is significantly more similar to a two-piece power-law distribution[6]. If we further slow down the crawling speed, the resulting snapshots contains a higher degree of edge distortion, and the derived degree distribution looks more similar to a single-piece power-law distribution, the result reported by earlier studies [2, 7]. *To a slow crawler, peers with long uptimes appear as high degree because many short-lived peers report them as neighbors. However, this is a mischaracterization since these short-lived peers are not all present at the same time. More importantly, this finding demonstrates that using distorted snapshots that are captured by slow crawlers can easily lead to incorrect characterizations of P2P overlays.*

Because we were unable to contact every top-level peer, the distribution in Figure 3(a) is biased slightly low since it does not include all edges. To address this problem, we split the data into Figures 3(b) and 3(c), which depict the neighbor degree distribution for reachable and unreachable peers, respectively. The data in Figure 3(b) is unbiased since we contacted each peer successfully, *i.e.*, we discovered every edge connected to these peers. The spike around a degree of 30 is more pronounced in this figure. Figure 3(c) presents the observed degree distribution for unreachable top-level peers (*i.e.*, overloaded or NATed). This distribution is biased low since we cannot observe the connections between pairs of these peers. In this data, a much greater fraction of peers have an observed degree below 30. Many of these peers probably have a true degree closer to 30, with the true distribution likely similar to that in Figure 3(b).

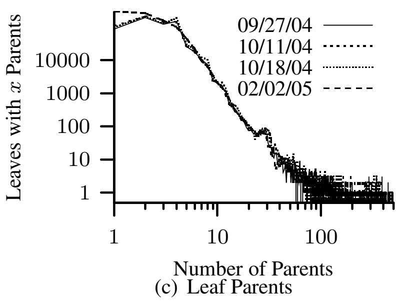
The degree distribution among contacted top-level peers has two distinct segments around a spike in degree of 30, resulting from LimeWire and BearShare's behavior of attempting to maintain 30 neighbors. The peers with higher degree represent other implementations that try to maintain a higher node degree or the rare user who has modified their client software. The peers with lower degree are peers which have not yet established 30 connections. In other words, the observed degree for these peers is temporary. They are in a state of flux, working on opening more connections to increase their degree. To verify this hypothesis, we plot the mean degree of peers as a function of their uptime in Figure 5. The standard deviation for these measurements is quite large (around $7-13$), indicating high



(a) Observed top-level degree distributions of a slow and a fast crawl



(b) Degree distribution from ultrapeers to leaves



(c) Leaf Parents

Figure 4: Different angles of degree distribution in Gnutella

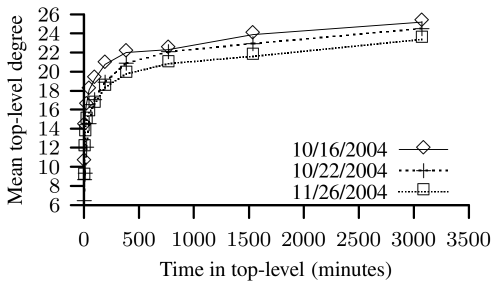


Figure 5: Mean degree as a function of uptime. Standard deviation is large (7–13).

variability. When peers first arrive, they quickly establish several connections. However, since node churn is high, they are constantly losing connections and establishing new ones. As time passes, long-lived peers gradually accumulate stable connections to other long-lived peers. We further explore this issue in Section 5 when we examine overlay dynamics.

Node Degree For Leaves: To characterize properties of the two-tier topology, we have examined the degree distribution between the top-level overlay and leaves, and vice versa. Figure 4(b) presents the degree distribution of connections from ultrapeers to leaf peers. Distinct spikes at 30, 45 and 75 degree are visible. The first two spikes are due to the corresponding parameters used in LimeWire and BearShare implementations, respectively. The third spike is due to a less common implementation. This figure shows that a significant minority of ultrapeers are connected to less than 30 leaf peers, which indicates availability in the system to accommodate more leaf peers.

In Figure 4(c), we present the degree of connectivity for leaf peers. This result reveals that most leaf peers connect to three ultrapeers or fewer (the behavior of LimeWire), a small fraction of leaves connect to several ultrapeers, and a few leaves ($< 0.02\%$) connect to an extremely large number of ultrapeers (100–3000).

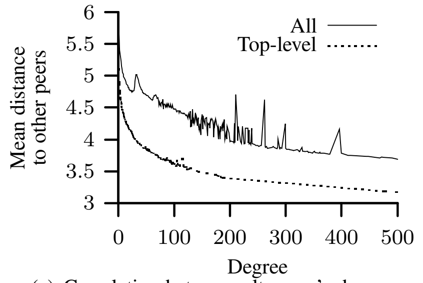
Implications of High Degree Peers: We observed a few outlier peers with an unusually high degree of connectivity in all degree distributions in this subsection. The main incentive for these peers is to reduce their mean distance to other peers. To quantify the benefit of this approach, Figure 6(a) presents the mean distance to other peers as a function of node degree, averaged across peers with the same degree. We show this for both the top-level overlay and across all peers. This figure shows that the mean path to participating peers exponentially decreases with degree. In other words, there are steeply diminishing returns from increasing degree as a way of decreasing distance to other peers.

Turning our attention to the effects of high-degree peers on the overlay, for scoped flood-based querying, the traffic these nodes must handle is proportional to their degree for leaves and proportional to the square of their degree for ultrapeers. Note that high-degree ultrapeers may not be able, or may not choose, to route all of the traffic between their neighbors. Thus, they may not actually provide as much connectivity as they appear to, affecting the performance of the overlay.

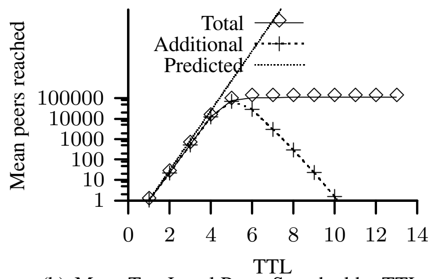
During our analysis, we discovered around 20 ultrapeers (all on the same /24 subnet) with an extremely high degree (between 2500 to 3500) in our snapshots. These high-degree peers are widely visible throughout the overlay, and thus receive a significant portion of exchanged queries among other peers. We directly connected to these high degree peers and found they do not actually forward any traffic[7]. We removed these inactive high degree peers from our snapshots when considering path lengths since their presence would artificially improve the apparent connectivity of the overlay.

4.2 Reachability

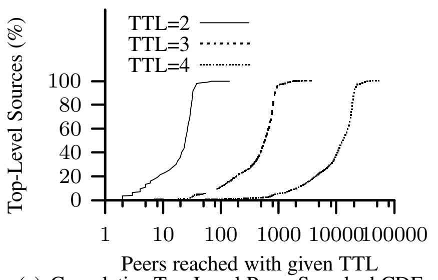
The degree distribution suggests the overlay topology might have a low diameter, given the moderately high degree of most peers. To explore the distances between peers in more detail, we examine two equally important properties of overlay topologies that express the reachability



(a) Correlation between ultrapeer's degree and its mean distance from other ultrapeers from the 10/18/04 snapshot

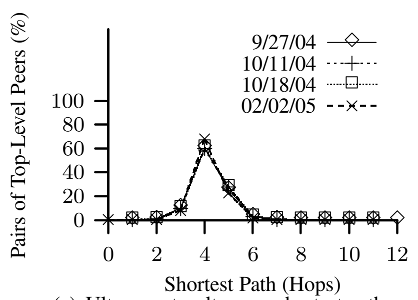


(b) Mean Top-Level Peers Searched by TTL from the 9/27/2004 snapshot

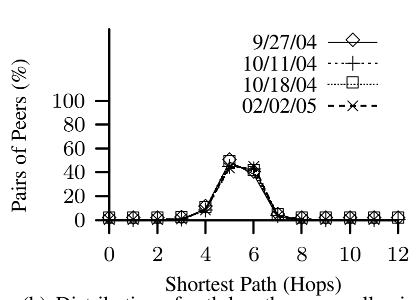


(c) Cumulative Top-Level Peers Searched CDF

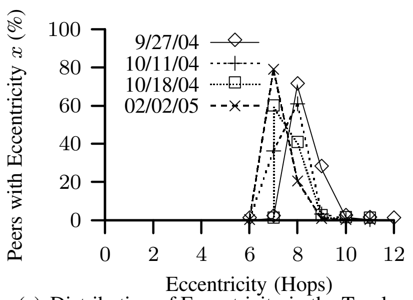
Figure 6: reachability, diameter, and shortest path in Gnutella topology



(a) Ultrapeer-to-ultrapeer shortest paths



(b) Distribution of path lengths across all pairs of peers



(c) Distribution of Eccentricity in the Top-level Overlay

Figure 7: Different angles on path lengths

of queries throughout the overlay: *(i)* the reachability of flood-based queries, and *(ii)* the pairwise distance between arbitrary pairs of peers.

Reachability of Flood-Based Query: Figure 6(b) depicts the *mean* number of newly visited peers and its cumulative value as a function of TTL, averaged across top-level peers in a single snapshot. The shape of this figure is similar to the result that was reported by Lv et al. (Figure 3 in [20]) which was captured in October 2000, with a significantly smaller number of peers (less than 5000). Both results indicate that the number of newly visited peers exponentially grows with increasing TTL up to a certain threshold and has diminishing returns afterwards. This illustrates that the dramatic growth of network size has been effectively balanced by the introduction of ultrapeers and an increase in node degree. Thus, while the network has changed in many ways, the percentage (but not absolute number) of newly reached peers per TTL has remained relatively stable. Figure 6(b) also shows the number of newly visited peers predicted by the Dynamic Querying formula (assuming a node degree of 30), which we presented in Section 2.1. This result indicates that the formula closely predicts the number of newly visited peers for TTL values less than 5. Beyond 5, the query has almost completely saturated the network.

Figure 6(c) shows a different angle of reachability for the same snapshot by presenting the Cumulative Distribution Function (CDF) of the number of visited peers from top-level peers for different TTL values. This figure shows the distribution of reachability for flood-based queries among participating peers. We use a logarithmic x-scale to magnify the left part of the figure for lower TTL values. The figure illustrates two interesting points: First, the total number of visited peers using a TTL of n is almost always an order of magnitude higher compared to using a TTL of $(n-1)$. In other words, TTL is the primary determinant of the mean number of newly visited peers independent of a peer's location. Second, the distribution of newly visited peers for each TTL is not uniform among all peers. As TTL increases, this distribution becomes more skewed (considering the logarithmic scale for x axis). This is a direct effect of node degree. More specifically, if a peer or one of its neighbors has a very high degree, its flood-based query reaches a proportionally larger number of peers.

Pair-wise Distance: Figure 7(a) shows the distribution of shortest-path lengths in terms of overlay hops among all pairs of top-level peers from four snapshots. Ripeanu et al. [23] presented a similar distribution for the shortest-path length based on snapshots that were collected between November 2000 and June 2001 with 30,000 peers. Comparing these results reveals two differences: *(i)* the pairwise path between peers over the modern Gnutella topology is *significantly more homogeneous in length, with shorter mean value* compared with a few years ago. More specifically, the old snapshot shows 40% and 50% of all paths having a length of 4 and 5 hops whereas our results show a surprising 60% of all paths having a length of 4. *(ii)* the results from our snapshots are nearly identical; whereas in [23], there is considerable variance from one crawl to another. In summary, *the path lengths have become shorter, more homogeneous, and more stable*.

Effect of Two-Tier Topology: To examine the effect of the two-tier overlay topology on path length, we also plot the path length between all peers (including leaves) in 7(b). If each leaf had only one ultrapeer, the distribution of path length between leaves would look just like the top-level path lengths (Figure 7(a)), but right-shifted by two. However, since each leaf peer has multiple parents, the path length distribution between leaves (and thus for all peers) has a more subtle relationship with Figure 7(a). Comparing Figures 7(a) and 7(b) shows us the cost introduced by using a two-tier overlay. In the top-level, most paths are of length 4. Among leaves, we see that around 50% of paths are of length 5 and the other 50% are of length 6. Thus, getting to and from the top-level overlay introduces an increase of 1 to 2 overlay hops.

Eccentricity: The longest observed path in these four snapshots was 12 hops, however the vast majority (99.5%) of paths have a length of 5 hops or less. To further explore the longest paths in the topology, we examined the distribution of eccentricity in the top-level overlay. The eccentricity of a peer is the distance from that peer to the most distant other peer. More formally, given the function $P(i, j)$ that returns the shortest path distance between nodes i and j, the eccentricity, E_i of node i is defined as follows: $E_i = \max(P(i, j),\ \forall j)$. Figure 7(c) shows the distribution of eccentricity in four topology snapshots. This figure shows that the distribution of eccentricity is rather homogeneous and low which is an indication that the overlay graph is a relatively balanced and well-connected mesh, rather than a chain of multiple groups of peers.

4.3 Small World

Recent studies have shown that many biological and man-made graphs (*e.g.*, collaborations among actors, the electrical grid, and the WWW graph) exhibit "small world" properties. In these graphs, the mean pairwise distance between nodes is small and nodes are highly clustered compared to random graphs with the same number of vertices and edges. A study by Jovanovic et al. [12] in November–December 2000 concluded that the Gnutella network exhibits small world properties as well. Our goal is to verify to what extent recent top-level topologies of the Gnutella network still exhibit small world properties despite growth in overlay population, an increase in node degree, and changes in overlay structure. The clustering coefficient of a graph, C_{actual}, represents how frequently each node's neighbors are also neighbors, and is defined as follows [35]:

Graph	L_{actual}	L_{random}	C_{actual}	C_{random}
New Gnutella	4.17–4.23	3.75	0.018	0.00038
Old Gnutella	3.30–4.42	3.66	0.02	0.002
Movie Actors	3.65	2.99	0.79	0.00027
Power Grid	18.7	12.4	0.08	0.005
C. Elegans	2.65	2.25	0.28	0.05

Table 3: Small World Characteristics

$$C(i) = \frac{D(i)}{D_{max}(i)}, \quad C_{actual} = \frac{\sum_i C(i)}{|V|}$$

$D(i)$, $D_{max}(i)$ and $|V|$ denote the number of edges between neighbors of node i, the maximum possible edges between neighbors of node i, and the number of vertices in the graph, respectively. For example, if node A has 3 neighbors, they could have at most 3 edges between them, so $D_{max}(A) = 3$. If only two of them are connected together, that's one edge and we have $D(A) = 1$ and $C(A) = \frac{1}{3}$. $C(i)$ is not defined for nodes with fewer than 2 neighbors. Thus, we simply exclude these nodes from the computation of C_{actual}. Table 3 presents ranges for the clustering coefficient (C_{actual}) and mean path length (L_{actual}) for the Gnutella snapshots from Table 1 as well as the mean values from four random graphs with the same number of vertices and edges (*i.e.*, C_{random} and L_{random}). Because computing the true mean path lengths (L_{random}) is computationally expensive for large graphs, we used the mean of 500 sample paths selected uniformly at random. We also include the information presented by Jovanovic et al. [12] and three classic small world graphs [35].

A graph is loosely identified as a small world when its mean path length is close to random graphs with the same number of edge and vertices, but its clustering coefficient is orders of magnitude larger than the corresponding random graph (*i.e.*, L_{actual} and L_{random} are close, but C_{actual} is orders of magnitude larger than C_{random}). All three classic small world graphs in the table exhibit variants of these conditions. Snapshots of modern Gnutella clearly satisfy these conditions which means that modern Gnutella still exhibits small world properties.

Comparing the clustering coefficient between modern Gnutella and old Gnutella shows that modern Gnutella has less clustering. A plausible explanation is the increased size, which provides the opportunity for more diverse connectivity to other peers. A high clustering coefficient implies a larger fraction of redundant messages in flood-based querying. The observed clustering could be a result of factors like peer bootstrapping, the peer discovery mechanism, and overlay dynamics. Further analysis is needed to better

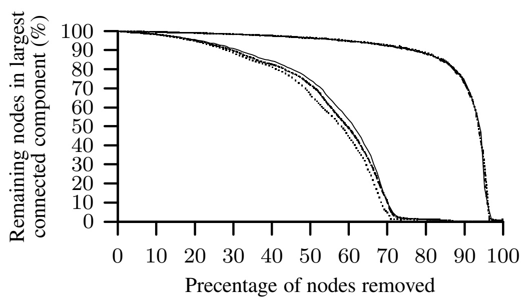


Figure 8: Fraction of remaining nodes in the largest connected component as a function of the percentage of original nodes removed for the 9/27, 10/11, and 10/18 snapshots. The top (overlapped) lines and the bottom three lines present random and pathological node removal scenarios, respectively.

understand the underlying causes. Section 5 shows how peer churn is one factor that contributes to clustering.

4.4 Resilience

We also examine the resilience in different snapshots of the Gnutella overlay topology using two different types of node removal: *(i)* random removal, and *(ii)* pathologically removing the highest-degree nodes first. An early study [24] conducted the same analysis on Gnutella based on a partial topology snapshot, finding that the overlay is resilient to random departures, but under pathological node removal quickly becomes very fragmented (after removing just 4% of nodes).

Figure 8 depicts the fraction of remaining nodes in the topology which remain still connected in both the random and pathological node removal. *This figure clearly shows the Gnutella overlay is not only extremely robust to random peer removals, but it also exhibits high resilience to pathological node removal.* Even after removing 85% of peers randomly, 90% of the remaining nodes are still connected. For the pathological case, after removing the 50% of peers with the highest-degree, 75% of the remaining nodes remain connected. There are two possible factors contributing to this difference with earlier results [24]: *(i)* the higher median node degree of most nodes in modern Gnutella, and *(ii)* a non-negligible number of missing nodes and edges in the partial snapshot of the earlier study. Our result implies that complex overlay construction algorithms (*e.g.*, [36]) are not always a necessary prerequisite for ensuring resilience in unstructured overlays.

5 Overlay Dynamics

In Section 4, we characterized the graph-related properties of individual snapshots of the overlay topology. However, in practice the overlay topology is inherently dynamic since connections (*i.e.*, edges) are constantly changing. These dynamics can significantly affect the main functionality of the overlay which is to provide connectivity and efficiently route the messages (*e.g.*, queries, responses) among participating peers. Characterizing overlay dynamics enables us to examine their impact on performance of P2P applications. For example, a query or response message can be routed differently or even dropped as a result of changes in the edges of the overlay. To our knowledge, aggregate dynamics of unstructured P2P overlay have not been studied. There are two basic causes for observed dynamics in the overlay topology as follows:

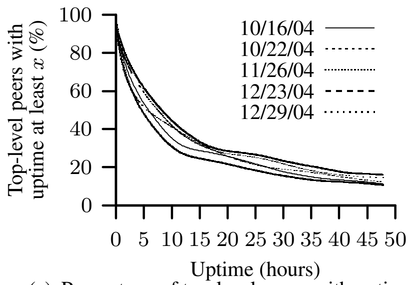
- Dynamics of Peer Participation: When a peer joins (or departs) the network, it establishes (or tears down) its connections to other participating peers in the overlay. Therefore, these changes in overlay edges are *user-driven*[8].

- Dynamics of Neighbor Selection: Two existing peers in the overlay may establish a new (or tear down an existing) connection between them. Such a change in edges is not triggered by users and thus considered *protocol-driven*.

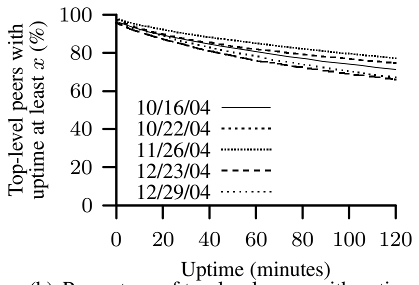
Note that the user-driven dynamics of peer participation are likely to exhibit similar heavy-tailed distributions in different P2P applications [31, 28]. Therefore, characterization of user-driven dynamics in the overlay provides a useful insight for design of other Gnutella-like unstructured P2P overlays.

In this section, we characterize the dynamics of the Gnutella network. More specifically, we want to investigate *(i) whether a subset of participating peers form a relatively stable core for the overlay, (ii) what properties (such as size, diameter, degree of connectivity or clustering) this stable core exhibits, and (iii) what underlying factors contribute to the formation and properties of such a stable core.*

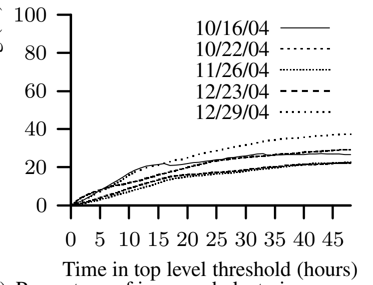
Methodology: Our main goal is to determine whether observed dynamics (*i.e.*, the rate of change in the edges of the overlay) are different at various regions of the overlay. We primarily focus on the top-level overlay in our analysis, because leaf nodes do not forward traffic and therefore do not provide meaningful connectivity between peers. One key issue is to define a core region for the "spaghetti-like" overlay. We use the following methodology to identify and characterize any potentially stable core for the overlay. Intuitively, if the overlay has a stable core, it must contain the long-lived peers of the overlay. Therefore, to identify the stable core of the overlay at any point of time, we select the subset of participating peers who have been part of the overlay for at least τ minutes, *i.e.*, all peers whose uptime is longer than a threshold τ. We call this subset of peers



(a) Percentage of top-level peers with uptime at least x

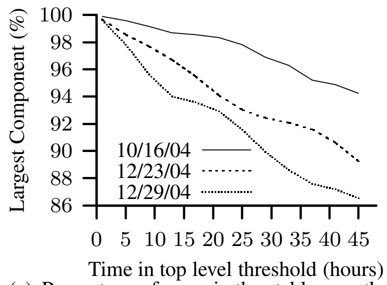


(b) Percentage of top-level peers with uptime at least x (zoomed in)

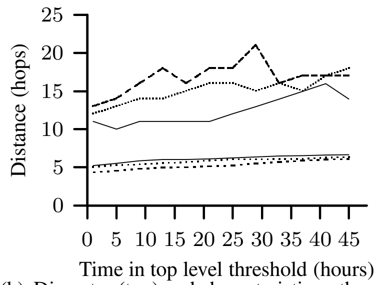


(c) Percentage of increased clustering among stable nodes, relative to a randomized topology for 5 different snapshots

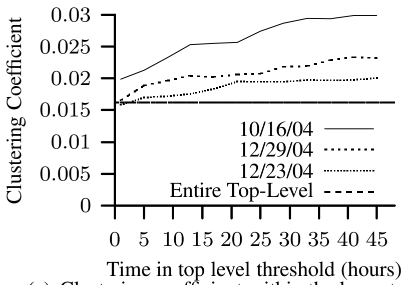
Figure 9: Number of stable peers and their external connectivity for different τ



(a) Percentage of peers in the stable core that are part of the core's largest connect component



(b) Diameter (top) and characteristic path length (bottom) of the largest connected component of the stable core



(c) Clustering coefficient within the largest connected component of the stable core

Figure 10: Different angles of connectivity with the stable core

the *stable peers*, or $SP(\tau)$, and only focus on this subset in our analysis. However, by changing τ, we can control the minimum uptime of selected peers and thus the relative stability and size of $SP(\tau)$.

To conduct this analysis, we use several slices of our dataset where each slice is a period of 48 hours of continuous back-to-back topology snapshots, with hundreds of snapshots per slice. Let's consider the last captured snapshot over each 48 hour period as a reference snapshot. Any peer in the reference snapshot must have joined the overlay either before or during our measurement period. By looking back through the snapshots, we can determine (with accuracy of a few minutes) the arrival time of all peers that joined during the measurement period. For those peers that were present for the entire measurement period, we can conclude that their uptime is at least 48 hours. Having this information, we can annotate all peers in the reference snapshot with their uptime information. Figure 9(a) depicts the CCDF of uptime among existing peers in the reference snapshot for several slices (Figure 9(b) presents the initial part of the same graph). In essence, this figure presents the distribution of uptime among participating peers in steady state, implying that the size of $SP(\tau)$ exponentially decreases with τ. This is more visible over longer time scales. Furthermore, this also implies that the total number of possible connections within $SP(\tau)$ dramatically decreases with τ.

Internal Connectivity Within the Stable Core: To study different angles of connectivity among ultrapeers within $SP(\tau)$, we focus only on the connections of the overlay where both end points are inside $SP(\tau)$, *i.e.*, we remove all edges to peers outside $SP(\tau)$. We call this the stable core overlay or $SC(\tau)$. The first question is: *whether $SC(\tau)$ is fully connected?* Figure 10(a) depicts the fraction of ultrapeers within $SC(\tau)$ that are in the largest connected component, as a function of τ. This figure clearly demonstrates that while the fraction of connected peers slightly decreases with τ over long times scales, a significant majority (86%–94%) of peers within $SC(\tau)$ remain fully connected. The minor drop in the percentage of connected peers is due to exponential decrease in number of peers within $SC(\tau)$, which in turn reduces the number of edges among peers, and thus affects the opportunity for pairwise connectivity. The second question is: *how clustered and dense is the connected portion of the core overlay?* Figure 10(b) shows the diameter and characteristic (mean) path length among fully connected peers in the stable core overlay. Interestingly, both the mean path length and the diameter of the stable core overlay remain relatively stable as τ increases, despite the dramatic drop in number of edges. Furthermore, the mean path length for the stable core overlay, even when it has a very small population (only 10% of top-level peers for τ=45h), is around 5 hops, very close to the mean path length for the entire top-level overlay (4.17–4.23 from the first row of Table 3). Finally, Figure 10(c) depicts the evolution of the clustering coefficient for the stable core overlay as τ increases, along with the clustering coefficient for the entire top-level overlay in the reference snapshot. This figure shows two important points: *(i)* peers within the stable core overlay are more clustered together than the entire top-level overlay on average, and, more importantly, *(ii)* connectivity among peers within the stable core overlay becomes increasingly more clustered with τ. This latter point implies that *the longer a peer remains in the overlay, the more likely it establishes connections to peers with equal or higher uptimes,* i.e.*, the more biased its connectivity becomes toward peers with higher uptime. Since connections for all participating peers exhibit the same behavior, connectivity of the overlay exhibits a biased "onion-like" layering where peers with similar uptime (a layer) have a tendency to be connected to peers with the same or higher uptime (internal layers of the onion).* Since the size of $SP(\tau)$ decreases with τ, this means that internal layers are both smaller and more clustered.

External Connectivity to/from the Stable Core: To quantify the connectivity between $SC(\tau)$ and the rest of the overlay we examined whether peers within $SC(\tau)$ have a higher tendency to connect to each other rather than peers outside the core. To quantify any potential tendency, we calculate the ratio of internal edges to the total number of edges and compare that with the same ratio for a randomly generated graph with the same number of nodes, same degree distribution among nodes, and same number of edges. For a fair comparison, we present the notion of a *half edge* for a graph as follows: we cut the edge E_{ij} between two nodes i and j, and define $HalfEdge(i,j)$ as the half of E_{ij} that is connected to node i. Then, the ratio of internal to total half-edges can be calculated as follows:

$$R = \frac{\sum_{i \in SC} \sum_{j \in SC} HalfEdge(i,j)}{\sum_{i \in SC} \sum_{\text{all} j} HalfEdge(i,j)}$$

Figure 9(c) depicts $(R_g - R_r)/R_r$ as a function of τ where R_g and R_r denote the value of R for several snapshots and their corresponding randomly generated graphs, respectively. This figure demonstrates that the longer a peer remains in the network, its connectivity becomes more biased towards peers with the same or higher uptime. This is another evidence that peers exhibit an onion-like biased connectivity and the degree of such bias increases with uptime.

Implications of Stable and Layered Core Overlay: The connectivity of the core overlay implies that all peers within the core do not depend on peers outside the core for reachability. In other words, the core overlay provides a stable and efficient backbone for the entire top-level overlay that ensures connectivity among all participating peers despite the high rate of dynamics among peers outside the core.

5.1 Examining Underlying Causes

A key question is: *how does this onion-like layered connectivity form in the overlay in an unintentional and uncoordinated fashion?* To address this issue, we quantify the contribution of user-driven and protocol-driven dynamics in changes of the edges of the overlay. We can distinguish protocol-driven versus user-driven changes in edges between two snapshots of the overlay as follows: if at least one of the endpoints for a changing edge has arrived (or departed) between two snapshots, that change is user-driven. Otherwise, a changing edge is considered protocol-driven. To answer the above question, we examine a 48-hour slice of back-to-back snapshots from 10/14/2004 to 10/16/2004, using the first snapshot as a reference. Given a slice, we can detect new or missing edges in any snapshot compared to the reference snapshot, for peers in both snapshots. Let δ_{p-} and δ_{u-} (δ_{p+} and δ_{u+}) denote the normalized ratio of missing (and new) edges in a snapshot due to protocol-driven (p) and user-driven (u) causes, normalized by the number of edges in the reference snapshot. Figure 11(a) and 11(b) depict δ_{-}=δ_{p-}+δ_{u-} and δ_{+}=δ_{p+}+δ_{u+} for back-to-back snapshots for the slice under investigation. Each figure also depicts the breakdown of changes in edges into two groups: protocol-driven and user-driven changes. Note that δ_p and δ_u are by definition cumulative. The left graph (δ_{-}) shows that around 20% and 30% of edges in the overlay are removed due to protocol-driven and user-driven factors during the first 100 minutes, respectively. After this period, almost all removed edges are due to departing peers. Similarly, from the right graph, many edges are added during the first 100 minutes due to both protocol-driven factors and the arrival of new peers. After this period, almost

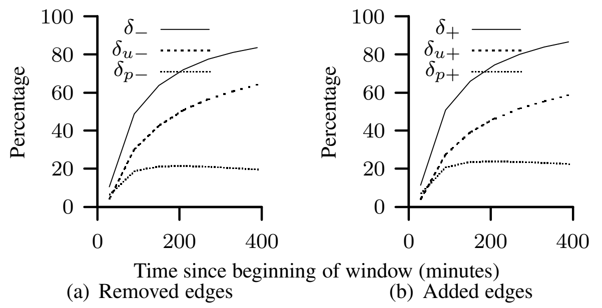


Figure 11: Contribution of user- and protocol-driven dynamics in variations of edges in the overlay

all new edges involve a newly arriving peer. These results shows two important points: First, each peer may establish and tear down many connections to other peers during the initial 100 minutes of its uptime. But peers with higher uptime (*i.e.*, peers inside $SC(\tau)$ for $\tau \geq 100$ min), maintain their connections to their remaining long-lived neighbors, and only add (or drop) connections to arriving (or departing) peers. This behavior appears to explain the formation of the biased onion-like layering in connectivity within the overlay. Second, user-driven dynamics are the dominant factor in long-term changes of the overlay. Since dynamics of peer participations exhibit similar dynamics in different P2P systems [31], other Gnutella-like overlays are likely to show similar behavior. We plan to conduct further investigations to better understand the underlying dynamics that contribute to this behavior.

6 Related Work

As listed throughout this paper, there are a handful of prior studies on characterizing peer-to-peer overlay topologies in file-sharing applications [23, 2, 20, 12]. These studies are more than three years old, did not verify the accuracy of their captured snapshots, and conducted only limited analysis. A recent study [18] used both passive measurement and active probing of 900 super nodes to study behavior of the Kaaza overlay. They have mostly focused on the number of observed connections (within the top-level overlay and from the top-level overlay to leaf nodes) and their evolution with time. However they have not examined detailed graph-related properties of the overlay, or collective dynamics of the entire overlay topology, both of which are investigated in this paper.

There has been a wealth of measurement research on other properties of peer-to-peer systems. These studies cover several topics: *(i)* file characteristics [6, 17, 3, 19], *(ii)* transfer characteristics [10, 17], *(iii)* peer characteristics [25, 24], *(vi)* query characteristics [26, 3, 16, 4], and *(v)* comparisons of different implementations [15, 11]. Since they explore different aspects of peer-to-peer networks, these studies complement our work. There have also been several modeling and simulation-based studies on improvement of search in Gnutella-like P2P networks [5, 38, 37, 27]. Our characterization can be directly used by these studies as a reference for comparison of suggested topology models, and our captured overlay snapshots can be used for trace-driven simulation of their proposed search mechanisms.

Finally, the research studies on characterization of the Internet topology (*e.g.*, [8]) and network topology generators (*e.g.*, [34]) are closely related to our work. However, these studies focus on the Internet topology rather than an overlay topology. We plan to conduct further characterization of the Gnutella topology by applying some of the suggested graph analysis in these studies to the Gnutella overlay topology.

7 Conclusions

In this paper, using Gnutella, we presented the first detailed characterization of an unstructured two-tier overlay topology that is typical of modern popular P2P systems, based on accurate and complete snapshots. We described fundamental challenges in capturing accurate snapshots, and demonstrated that inaccurate snapshots can lead to erroneous conclusions—such as a power-law degree distribution. We characterized the graph-related properties of individual snapshots, the dynamics of the overlay topology across different time scales, and investigated the underlying causes and implications. Our main findings are summarized in Section 1.1.

This study developed essential insights into the behavior of overlay topologies which are necessary to improve the design and evaluation of peer-to-peer file-sharing applications. The existence of a stable well-connected core of long-lived peers suggests that there may be benefits in terms of increasing search resilience in the face ofd the overlay dynamics, by biasing/directing the search towards longer lived peers and therefore towards this core. It may also be useful to cache indexes or content at long-lived peers in order to reduce load on the stable core, especially if the biased forwarding of queries is adopted. For example, the idea of one-hop replication [21], intended for power-law topologies, can be changed to a probabilistic one-hop replication biased towards peers with longer uptime.

We are continuing this work along a number of directions. We are actively monitoring the Gnutella network and plan to further examine the dynamics of peer participation over short time scales, explore any longer term trends in the topology, and observe variations in several key properties (*e.g.*, small-world coefficient, degree distribution, and mean pairwise distance) with time. We are applying our techniques to develop characterizations of the eDonkey/Overnet and BitTorrent P2P networks in ongoing work.

References

[1] slyck.com. http://www.slyck.com, 2005.

[2] L. A. Adamic, R. M. Lukose, B. Huberman, and A. R. Puniyani. Search in Power-Law Networks. *Physical Review E*, 64(46135), 2001.

[3] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oct. 2000.

[4] F. S. Annexstein, K. A. Berman, and M. A. Jovanovic. Latency effects on reachability in large-scale peer-to-peer networks. In *Symposium on Parallel Algorithms and Architectures*, pages 84–92, 2001.

[5] Y. Chawathe, S. Ratnasamy, and L. Breslau. Making Gnutella-like P2P Systems Scalable. In *SIGCOMM*, 2003.

[6] J. Chu, K. Labonte, and B. N. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *ITCom: Scalability and Traffic Control in IP Networks II Conferences*, July 2002.

[7] clip2.com. Gnutella: To the Bandwidth Barrier and Beyond, Nov. 2000.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *SIGCOMM*, 1999.

[9] A. Fisk. Gnutella Dynamic Query Protocol v0.1. Gnutella Developer's Forum, May 2003.

[10] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *SOSP*, 2003.

[11] Q. He and M. Ammar. Congestion Control and Message Loss in Gnutella Networks. In *Multimedia Computing and Networking*, Jan. 2004.

[12] M. Jovanovic, F. Annexstein, and K. Berman. Modeling Peer-to-Peer Network Topologies through "Small-World" Models and Power Laws. In *TELFOR*, Nov. 2001.

[13] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *Globecom*, Nov. 2004.

[14] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport Layer Identification of P2P Traffic. In *International Measurement Conference*, Oct. 2004.

[15] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, and E. Zegura. Bootstrapping in Gnutella: A Measurement Study. In *PAM*, Apr. 2004.

[16] A. Klemm, C. Lindemann, M. Vernon, and O. P. Waldhorst. Characterizing the Query Behavior in Peer-to-Peer File Sharing Systems. In *Internet Measurement Conference*, Oct. 2004.

[17] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa Network. In *WIAPP*, 2003.

[18] J. Liang, R. Kumar, and K. W. Ross. The KaZaA Overlay: A Measurement Study. *Computer Networks Journal (Elsevier)*, 2005.

[19] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in P2P File Sharing Systems. In *INFOCOM*, Mar. 2005.

[20] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *International Conference on Supercomputing*, 2002.

[21] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make Gnutella scalable? In *IPTPS*, 2002.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[23] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing Journal*, 6(1), 2002.

[24] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems Journal*, 8(5), Nov. 2002.

[25] S. Sen and J. Wang. Analyzing Peer-To-Peer Traffic Across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, Apr. 2004.

[26] K. Sripanidkulchai. The popularity of Gnutella queries and its implications on scalability. http://www-2.cs.cmu.edu/ kunwadee/research/p2p/paper.html, Jan. 2001.

[27] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM*, 2003.

[28] K. Sripanidkulchai, B. Maggs, and H. Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Internet Measurement Conference*, Oct. 2004.

[29] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 2002.

[30] D. Stutzbach and R. Rejaie. Capturing Accurate Snapshots of the Gnutella Network. In *Global Internet Symposium*, pages 127–132, Mar. 2005.

[31] D. Stutzbach and R. Rejaie. Characterizing Churn in Peer-to-Peer Networks. Technical Report 2005-03, University of Oregon, May 2005.

[32] D. Stutzbach and R. Rejaie. Characterizing the Two-Tier Gnutella Topology. In *SIGMETRICS*, Extended Abstract, June 2005.

[33] D. Stutzbach and R. Rejaie. Evaluating the Accuracy of Captured Snapshots by Peer-to-Peer Crawlers. In *Passive and Active Measurement Workshop*, Extended Abstract, pages 353–357, Mar. 2005.

[34] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network Topology Generators: Degree-Based vs. Structural. In *SIGCOMM*, 2002.

[35] D. J. Watts. Six Degrees. In *The Essence of a Connected Edge*. ACM Press, 2003.

[36] R. H. Wouhaybi and A. T. Campbell. Phenix: Supporting Resilient Low-Diameter Peer-to-Peer Topologies. In *INFOCOM*, 2004.

[37] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *International Conference on Data Engineering*, Mar. 2003.

[38] B. Yang, P. Vinograd, and H. Garcia-Molina. Evaluating GUESS and Non-Forwarding Peer-to-Peer Search. In *IEEE International Conference on Distributed Systems*, 2004.

Notes

[1]An earlier version of our work on graph-related properties of Gnutella appeared as an extended abstract in SIGMETRICS 2005 [32].

[2]Throughout this paper, by "uptime" we mean the time that has elapsed since the peer has arrived.

[3]http://netflow.internet2.edu/weekly/

[4]The degree distribution for all the presented results is limited to 500, which includes all but a handful of peers with larger degree that are discussed later.

[5]To reduce the crawling speed, we simply limited the degree of concurrency (*i.e.*, number of parallel connections) to 60 in Cruiser.

[6]To properly compare these snapshots with different sizes, the y-axis in Figure 4(a) was normalized by number of peers in the snapshot

[7]To our surprise, it appears that these peers monitor exchanged messages among other participating peers. They could be trying to locate copyright infringement among Gnutella users or collecting ratings information to measure which songs consumers might like to buy.

[8]Note that Gnutella does not run as a daemon. Therefore, peer arrival/departure is a reliable indication of user action. We are mindful that dynamic IP addresses could force some peers to leave and rejoin the network with a new address. However, this does not affect our analysis since we examine the effect of each departure/arrival event on the overlay dynamics.

Should Internet Service Providers Fear Peer-Assisted Content Distribution?

Thomas Karagiannis
U.C. Riverside

Pablo Rodriguez
Microsoft Research

Konstantina Papagiannaki
Intel Research Cambridge

Abstract

Recently, peer-to-peer (P2P) networks have emerged as an attractive solution to enable large-scale content distribution without requiring major infrastructure investments. While such P2P solutions appear highly beneficial for content providers and end-users, there seems to be a growing concern among Internet Service Providers (ISPs) that now need to support the distribution cost. In this work, we explore the potential impact of future P2P file delivery mechanisms as seen from three different perspectives: i) the content provider, ii) the ISPs, and iii) individual content consumers. Using a diverse set of measurements including BitTorrent tracker logs and payload packet traces collected at the edge of a 20,000 user access network, we quantify the impact of peer-assisted file delivery on end-user experience and resource consumption. We further compare it with the performance expected from traditional distribution mechanisms based on large server farms and Content Distribution Networks (CDNs).

While existing P2P content distribution solutions may provide significant benefits for content providers and end-consumers in terms of cost and performance, our results demonstrate that they have an adverse impact on ISPs' costs by shifting the associated capacity requirements from the content providers and CDNs to the ISPs themselves. Further, we highlight how simple "locality-aware" P2P delivery solutions can significantly alleviate the induced cost at the ISPs, while providing an overall performance that approximates that of a perfect world-wide caching infrastructure.

1 Introduction

Peer-to-peer (P2P) networks, where commodity personal computers form a cooperative network and share their resources (storage, CPU, bandwidth), have recently emerged as a solution to large scale content distribution without requiring major infrastructure investments. By capitalizing on the bandwidth of end-nodes, content providers that use peer-assisted solutions can benefit from a cost-effective distribution of bandwidth-intensive content to thousands of simultaneous users, both Internet-wide and in private networks.

Peer-assisted solutions are inherently self scalable, in that the bandwidth capacity of the system increases as more nodes arrive: each new node requests service from, but also provides service to, the other nodes. The network can thus spontaneously adapt to the demand by taking advantage of the resources provided by every end-node, thus making it more resilient to "flash crowd" events, which may challenge content distribution networks with hundreds of servers [10]. Overall, the system's capacity grows at the same rate as the demand, creating limitless scalability for a fixed cost.

The best example of such peer-assisted content distribution architectures is BitTorrent, which has been embraced by several content providers (Lindows, Blizzard) to reduce the load from congested servers, minimize the distribution cost, and improve download times of software and patch releases. However, while such peer-assisted architectures can provide significant benefits to end-users and content providers, there seems to be a growing concern among Internet Service Providers (ISPs) regarding the cost of supporting such solutions. Since demand is shifted from data centers to end-nodes, peers become servers for other peers at the network edge. This shift increases the amount of data served by each ISP without a corresponding increase in revenue from the peer-hosted data services provided.

In this paper, we explore the potential impact of future peer-assisted mechanisms as seen from three different perspectives: i) the content provider, ii) the ISPs, and iii) the individual users. In particular, we focus on how peer-assisted solutions affect ISP's traffic as load is shifted from the content provider's data center to peers at the edge of the network. We study how peer-assisted solutions affect ISPs over a range of parameters, and compare them with other solutions such as deploying large server farms or caching-based solutions. To this extent, we use a diverse set of measurements including BitTorrent tracker logs and payload packet traces collected on the Internet link of an ISP network.

Despite the benefits for content providers and end-consumers, our results demonstrate that peer-assisted content distribution solutions have an adverse impact on ISPs' costs. In particular, we show that current peer-assisted solutions roughly double the total traffic on the ISPs access link as well as their peak load due to the outbound network traffic. The reason is the lack of consideration of peer-assisted algorithms toward optimizing ISP's bandwidth. Such overhead is pushing a number of ISPs toward regulating such traffic, e.g. placing downloading caps. On the other hand, an ISP-friendly protocol that would minimize the ISPs' cost could ease such concerns and prevent providers from blocking or shaping P2P exchanges.

One way of providing an ISP-friendly system is by deploying caches that store the files being requested by the peers. Such a cache system would significantly reduce external network traffic for ISPs. However, caching infrastructures need to be compatible with a wide variety of P2P implementations and require extra hardware and maintenance support. Instead, we consider the possibility of adding small changes to existing peer-assisted distribution algorithms to mimic the performance of a caching solution. In this regard, we highlight how simple "locality-aware" peer-assisted delivery solutions can significantly alleviate the induced cost at the ISPs, while providing an overall performance that approximates that of a world-wide caching infrastructure.

The highlights of our work can be summarized in the following points:

- We provide a detailed study that sheds light on and quantifies the impact of peer-assisted content distribution solutions on ISPs based on real Internet traces.
- We present evidence that establish the potential for locality-aware "peer-assisted" solutions. We estimate and quantify file-availability and user-overlap in time where such solutions are feasible.
- We describe easily deployable architectures for efficient peer-assisted content distribution. For each case, we quantify the benefits and highlight potential savings.

Overall, our work aims at providing ISPs and content providers with a pragmatic, empirical cost-benefit analysis of current and future possible peer-assisted solutions for content distribution.

The remainder of the paper is structured as follows. In Section 2 we make the case for BitTorrent-like systems as P2P-mechanisms that could facilitate large scale content distribution and describe BitTorrent's functionality. In section 3, we demonstrate that current P2P systems and specifically BitTorrent do not exploit locality and present the implications of such a practice on a small ISP. In Section 4 we quantify the impact of P2P content distribution on the capacity requirements of an ISP network. Our findings motivate the need for a locality aware mechanism that directs peers to obtain the requested content from other peers located inside the same network, if they exist. We describe such mechanisms and evaluate their performance in section 5. In section 6, we describe related work. We discuss implications of our findings in section 7 and finally conclude in Section 8.

2 P2P as a mechanism for large scale content distribution

The P2P paradigm appears as an attractive alternative mechanism for large scale distribution of legal content (e.g., *Avalanche* [7]). Traditional content distribution mechanisms typically require vast investments in terms of infrastructure, usually in the form of CDNs and/or large server farms.

However, P2P content distribution is only viable by satisfying the requirements of both the content providers and the end-users.Any newly adopted mechanism should not incur additional overhead on any of the involved parties, i.e. the content provider, the users, and the users' ISPs.

We believe that BitTorrent-like P2P systems present a unique potential in achieving the aforementioned goals. As a P2P protocol, BitTorrent enjoys the benefits of a distributed system that is inherently more robust to events such as flash crowds, shown to be challenging even for CDNs with hundreds of servers [10], as well as cheaper in terms of infrastructure cost on the part of the content provider. Content distribution using BitTorrent has been shown to offer outstanding performance in terms of content delivery rates to the clients [22, 9]. Lastly, BitTorrent features a unique policy across P2P protocols, called the tit-for-tat policy, which is described below and ensures higher content availability.

In detail, the BitTorrent file distribution protocol specifies the functionality of three main entities [28]: a) a *tracker* which acts as a centralized server by coordinating the distribution and receiving information from all connected peers (over HTTP), b) a *torrent* meta-info file with basic description of the specific file (length, hash, name, etc.) and the tracker (IP), and c) the actual peers downloading the file. Peers only serving the file are referred to as "seeds", while downloading peers are called "leechers". Peers connect first to the tracker requesting a list of available peers in the network and then randomly select a subset of them to download from. This "peer" subset is periodically updated based on the contributions of each individual uploader with each peer trying to achieve the maximum available throughput. BitTorrent is characterized by a "choke/unchoke" mechanism that encourages cooperation among peers (peers upload more to the peers that offer them more data, *tit-for-tat* policy).

The "tit-for-tat" policy is a distinctive advantage of the BitTorrent system which renders it ideal for a P2P content distribution scheme: Peers are forced to always share the content during their downloads while "free-riders" [2] are indirectly banned from the network. Given that unlike today's P2P file-sharing networks, there is no notion of a "community" from the content distribution perspective, the tit-for-tat incentive ensures availability of the content as long as peers are requesting it.

However, BitTorrent users have no incentive of sharing the content once the download is complete. This practice is in contrast to the majority of P2P file-sharing networks where files are often made available even long after the completion of the download. Thus, availability may be compromised in a P2P content distribution scheme if demand is not high enough to ensure a sufficient number of active users. In the latter case, the provider will then have to ensure a larger number of active "seeds" in the network increasing its cost but facilitating availability. Even in this case, the benefits of a peer-assisted solution outweigh the cost compared to traditional content distribution approaches.

In the following sections we study both the effect of peer-assisted content distribution on the resource requirements of ISPs as well as the performance experienced by end-users. We use two different types of real measurement data. First, we study BitTorrent traffic from payload packet traces collected at a 20,000 user access network. These traces provide the view of file availability and potential savings from an edge-network view at small time scales. In addition, we analyze the tracker log for the RedHat v9.0 distribution that spans five months and offers a global perspective to the same problem at larger time scales.

3 P2P Content Distribution: the view from an edge network

In this section, we first look into the overhead of content distribution through BitTorrent as experienced by an edge network. We then quantify the savings that could be gained in the same scenario if locality was exploited. We conclude by examining the implications of locality-aware mechanisms on the user experience.

For this study we use three day-long packet traces collected with the high speed monitoring box described in [17] (Table 1). The monitor is installed on the access-link (full-duplex Gigabit Ethernet link) of a residential university that hosts numerous academic, research, and residential complexes with an approximate population of 20,000 users (a population equivalent of a small ISP). Our monitors capture the TCP/IP header and adequate payload information from all packets crossing the link in both directions to enable the identification of specific applications. P2P traffic accounts for approximately a third of the traffic in each one of the three traces (identified by the methodology described in [11]), while 13%-15% of the total traffic (8%-11% of the total packets) in the link is due to BitTorrent flows. The large fraction of BitTorrent traffic reveals its growing popularity and offers a sufficient sample to study its dynamics.

3.1 Methodology

Studying the dynamics of the BitTorrent network in the traces involves identifying all BitTorrent packets, determining their specific format and reconstructing all interactions across all BitTorrent flows. To identify BitTorrent flows and messages, we have developed a BitTorrent protocol dissector. While the *Ethereal* protocol analyzer [6] has a BitTorrent module, we encountered several problems with missed packets (e.g., TCP/IP packets that contained BitTorrent messages in the payload but were not identified by Ethereal), in the handling of out-of-order and fragmented BitTorrent messages and with multiple messages in the same TCP/IP packet. Furthermore, Ethereal's BitTorrent module cannot dissect tracker HTTP request/responses.

Specifically, we need to identify two types of messages for all BitTorrent flows: the tracker request/responses and the messages between peers. The nominal format for all packets can be found in [28]. The tracker request consists -among other things- of the peer id, the file hash and the local IP of the BitTorrent client (optional). The tracker response is usually a list of available clients for the requested file with statistics regarding the number of seeds, leechers etc. Regarding peer interactions, detecting the following BitTorrent messages is crucial to our analysis:

- *Handshake:* The first BitTorrent message transmitted by the initiator of the connection. It specifies the hash of the file and the *peer id*.

- *Piece:* BitTorrent files are divided into *Pieces*. The size of each piece is usually $262KB - 1MB$ and pieces are further subdivided in *blocks* of typically $16KB$. Blocks constitute the byte-segments that are actually transferred. The *Piece* message contains a data block of a given piece. The first nine bytes of the message specify the piece index, the byte offset within the piece and the size of the block.

- *BitField:* BitField specifies which pieces of the file are available for upload. It is a sequence of bits where set bits correspond to available pieces. It is typically the first message after the handshake.

- *Have:* The *Have* message advertises that the sender has downloaded a piece and the piece is now available for upload.

Table 1: Description of full-payload packet traces

Set	Date	Day	Start	Dur	Direc.	Src.IP	Dst.IP	Packets	Bytes	Aver.Util.	Aver. Flows/5min.
Jan	2004-01-20	Tue	16:50	24.6 h	Bi-dir.	2709 K	2626 K	2308 M	1223 G	110.5 Mbps	596 K
Apr	2004-04-23	Fri	15:40	33.6 h	Bi-dir.	4502 K	5742 K	3402 M	1652 G	109.4 Mbps	570 K
May	2004-05-19	Wed	07:50	28.6 h	Bi-dir.	1246 K	1301 K	3073 M	1706 G	132.5 Mbps	799 K

BitTorrent flows and messages are identified then through the following steps: a) individual packets are classified into flows based on the 5-tuple (source/destination IPs, source/destination TCP ports and protocol), and b) our dissector looks for the BitTorrent handshake message in the first two data packets of a flow (the BitTorrent handshake should be the first *data* packet of a BitTorrent flow but we allow for malformed packets). If the packet contains a BitTorrent handshake then the flow is flagged and all packets are examined by our dissector. All packets of flagged flows are dissected by keeping state of the interactions between the source and destination IPs. Note that the file transferred between the two peers of a BitTorrent flow is only specified at the *Handshake* message. Thus, while identifying a BitTorrent flow without the handshake message is possible, distinguishing the file transferred is not. In our case, this limitation only affects BitTorrent flows that are already active at the beginning of our traces.

Information on all BitTorrent interactions among peers allows us to track user requests, the associated downloads, the amount of time users request the same content at the same time, as well as the potential impact of locality aware peer-assisted content distribution on the utilization of network resources. We will look into each one of these metrics later in this section.

Identifying individual peers per file: Our analysis depends on robust identification of distinct peers in the BitTorrent network. However, identifying individual peers from BitTorrent messages is far from straightforward.

We can identify individual peers using a) tracker request and b) peer handshake messages. Limitations exist in both cases; while some of these restrictions are common, others are specific to each type of message. Thus each individual mechanism may be used to reinforce the accuracy of the other. Consequently, our peer identification relies on both methods. Pitfalls that need to be taken into account are the following :

Network Address Translators (NATs): Peers cannot be identified based solely on the IP address because of NATs, in which multiple peers appear to have the same IP address. To overcome this limitation we couple the IP with the observed *peer id*.

Peer id: The peer id is not unique for the same peer and varies with time and across flows. Typically the peer id comprises two parts: a) the first $6-8$ bytes of the peer id (out of 20 total) reveal the user client and version of the client (e.g, -AZ2202-, Azureus client, version 2.2.0.2), and b) random bytes. The random portion of the peer id changes with time and across flows. Thus, coupling the IP with the peer id, may result in double-counting peers if the random part varies. To avoid this pitfall, we only couple the IP with the *non-random* part of the id (the peer can be safely assumed to use the same BitTorrent client within the time scales of interest - in the rare case where multiple NATed users use the same client and IP address for the *same* file, we will consider them as one, thus underestimating locality which is the main theme of this work). Thus, *a distinct peer is now defined by the IP and non-random portion of the peer id*. Note that the number of bytes describing the non-random portion of the peer id varies with the client (e.g., for Azureus is 8 bytes while for BitComet is 6, etc.) [28].

BitSpirit (BS) client: Even the aforementioned definition of a peer does not guarantee identification of distinct peers due to the peer id assignment algorithm of the *BS* BitTorrent client. *BS* clients employ a function called "Adjust Client Identify", that modifies the non-random portion of the peer id to match the other peer's client in every flow! The BitTorrent client of other peers is known through the tracker responses (tracker responses include a list of peer IPs, ports and peer ids). On the contrary, the random part of the peer id remains constant across flows. This operation of *BS* clients is only specific to peer handshake messages and can be overcome by collapsing all different peer ids from the same IP which present the same random part of peer id into one user. Also, this restriction may be overcome by correlating peers as found by peer handshake messages with those shown by tracker messages for the specific IPs.

Proxies: The source IP of a tracker request does not always correspond to the IP of the peer even when the peer is not behind a NAT. A number of tracker requests are intercepted by *proxy* servers (tracker requests use HTTP) substituting the source IP of the peer with the one of the proxy. We identify such cases by the *proxy_fwd_for* header field (when available) which also reveals the original IP of the TCP packet. To avoid treating proxies as peers, we replace the proxy IP with the IP specified in the *proxy_fwd_for* field.

Random peer ids: A number of clients assigns random peer ids. This case affects mainly tracker requests and reports where the peer id may vary with time. As with the *BS* client restriction, correlating peer ids from handshake and tracker messages disambiguates individual users.

To achieve robust identification of distinct peers, we employ the above methodology in both types of messages (handshakes and tracker requests) separately and compare their outcomes. We found that the agreement between the two cases was sufficient to discriminate individual users per file. In total, we observed only 3 files out of a total of 360, where the produced list of users per file was not the same across the two methods. In these cases, we selected the list with the least number of users to avoid overestimating potential benefits of locality.

3.2 Hit Ratios

We quantify locality in terms of hit ratios. The hit ratio (analogous to caching hit ratio) refers to content that has already been downloaded and is present locally within our monitored ISP.

We examine the hit ratio along three dimensions:

File hit ratio, where we assume that the complete file is "cached" locally after the first download. Local caching would be the equivalent of a local-aware P2P system, where once a full copy exists within the ISP (either one peer has the full copy, or pieces of the file are spread across the ISP's customers), requests are served locally (assuming always active peers). Thus, the file hit ratio reveals the fraction of multiple downloads of the same content for the ISP in terms of the total number of downloaded files. Let N_i, be the user population for file i within the ISP with n being the total number of files. Then, the file hit ratio is defined as follows:

$$Hit\ Ratio = \frac{\sum_{i=1,n}(N_i - 1)}{\sum_{i=1,n} N_i}$$

Byte hit ratio, where we incorporate the file size in the hit ratio. The byte hit ratio is defined by multiplying each term in the sum of both the numerator and the denominator in the previous fraction by the size of file i (the file size can be inferred by the bitfield and piece messages[1]).

Piece hit ratio, where we examine what fraction of the incoming downloaded pieces for each file existed locally at the time of the download. Local pieces can be inferred by outgoing BitTorrent messages. Thus, there is a "Hit" for a downloaded piece, if the specific piece was advertised earlier in the trace by a local user through the *BitField* or *Have* messages. The total hit ratio is then the fraction of hits divided by the total number of downloaded pieces. Note that while our monitoring point disallows the capture of local interactions (packets transferred within the ISP boundaries not crossing our monitored link), our view of the status of each file is not limited for two reasons: a) the BitField message reflects the current status of the file with every new flow for the same file (BitTorrent protocol is characterized by a large number of open connections which yields a significant number of flows), and b) once a new piece is acquired, the peer advertises the specific piece to all its connected peers with the *Have* message.

[1]FileSize = Number of bits in the *BitField* × (maximum byte offset in a *Piece* message + block size of the *Piece* message)

Table 2: File, byte and piece hit ratios for the three traces.

	January	April	May
File Hit Ratio	14%	10.4%	18.2%
Byte Hit Ratio	12%	9.6%	13%
Piece Hit Ratio	6%	6%	11.8%

Table 2 presents the three hit ratios for all files across our three traces. The file hit ratio ranges from 10%-18% with the byte hit ratio being slightly lower. Taking into account specific piece and timing information reduces the hit ratio even more (6%-12%). Given the short duration of the traces (one day) and the small size of the monitored ISP the hit ratio is non negligible. Similar observation for one-day file and byte hit ratios have also been presented in [8][24] for the Kazaa network.

Requested files in the network are short-lived in accordance with previous findings in other P2P networks [25]. Only three files existed in both the April and May traces, while the January and April traces had only one file in common. Short-lived files imply that a P2P caching solution would not require large amounts of space and that if P2P nodes were to make their files available for a short period of time after the download is over, nodes could enjoy most of the possible sharing benefits.

3.3 Peer overlap in time

BitTorrent-like P2P systems assume the existence of a large number of *active* end users for a single file; peers participate in the sharing of the content by uploading at the same rate approximately as they download (tit-for-tat). The assumption of simultaneous active users, while valid globally, needs to also be valid within the boundaries of individual ISPs so that locality is beneficial. Note that an "active" peer in BitTorrent implies that the peer is currently downloading/uploading the specific content and not just participating in the network.

We quantify user overlap in time by tracking peer dynamics for all files in the network. We are interested in files that are downloaded/uploaded by at least two local peers throughout our traces, since locality or caching would have no impact on files requested by a single peer. Files with at least two users account for 10.5% (18/172), 8.7% (30/346) and 11.1% (34/306) of the files for our January, April and May traces respectively.

Peer overlap in time ranges from 30%-70% in our traces and is defined as the time during which more than one active users exist for at least one file versus the total time of the trace. Fig. 1 presents time overlap of peers for the April trace. The top line shows the number of "active" files in time; by active, we refer to files for which we observe activity at the specific time instance (download or upload). Note that we only plot files with more than one request over the whole trace. The bottom line shows the number of files with more than one "active" user at each time instance. Ac-

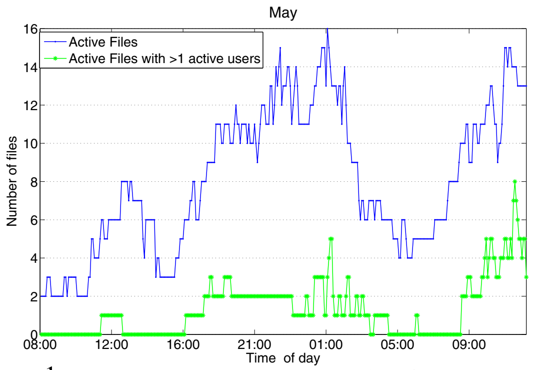


Figure 1: Peer overlap in time. The lines reflect files with at least two users over the whole trace. The top line presents active files in time. The bottom line shows the number of files with at least 2 active users in time. Approximately 60% of the time multiple active users coexist and could cooperate in the distribution of the content.

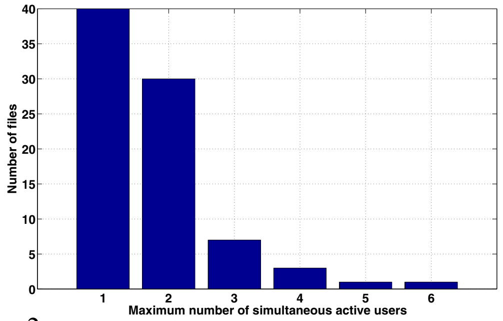


Figure 2: Histogram of the maximum number of simultaneous users per file for all files with at least two users in our traces. User-overlap is present for approximately 50% of the files, while 15% of the files have at least 3 simultaneous users at some point in time.

tive files and users follow the known diurnal patterns while towards the beginning of the second day the number of active users and files increases significantly. Overall, there exists at least one file with at least two active users 25%, 42% and 60% of the time for the January, April and May traces. Accounting for the fact that a number of already active BitTorrent flows at the beginning of our traces may have been missed, the percentages increase by 5%-10% after removing the first 5-10 hours of the traces.

The maximum number of simultaneous active peers per file in our traces is six. Fig. 2 shows the histogram of the maximum number of active users per file for all files requested by at least two peers. Locality could be exploited for 50% of the files where users coincide (we consider time overlap of at least 10 minutes to regard peers as simultaneous). Moreover, we observe at least three simultaneous peers for roughly 15% of the files.

3.4 Potential Savings on ISP bandwidth

Having established the co-existence of active users for the same file, we now quantify the percentage of "unnecessary" downloaded bytes. To estimate potential savings, we assume two scenarios: a) the caching case, where all local pieces are available once downloaded irrespective of the availability of the peer having the piece, and b) the peer-assisted case, where only local pieces in active users are

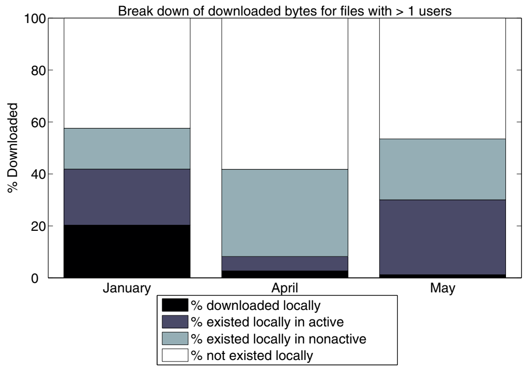


Figure 3: Breakdown of downloaded bytes for files with at least 2 active users. 70%-90% of existing local pieces are downloaded externally while 50%-90% of these pieces exist in active peers.

considered. Local pieces are inferred by the *BitField* and *Have* messages as discussed previously.

70%-90% of existing local pieces and 50%-90% of existing local pieces in active users are downloaded externally! Fig. 3 summarizes our findings by presenting a breakdown of all downloaded bytes for files with $N_i > 1$. Only a minimum portion of bytes is downloaded locally even though more than 20% of the bytes exist in active users (with the exception of April). In an ideal caching scenario, at least 40% of the content exists and could be downloaded locally.

3.5 Performance Implications for the User

Locality-aware peer-assisted content distribution mechanisms may have further benefits in terms of performance for individual peers. First, edge networks may feature much wider bottlenecks than the global Internet. In addition, the number of hops between peers is likely to be smaller and the associated propagation delays shorter, if the traffic stays within the ISP. For instance, a Gigabit Ethernet Local Area Network is likely to offer shorter, higher-throughput paths, to local clients compared to the case where clients are redirected to cross the Internet in order to retrieve the same content. To test this assumption we proceed as follows.

Our packet traces allow us to observe the throughput obtained by each user for each file retrieval at each 5 minute interval. The aforementioned throughput value corresponds to the performance experienced by the user using today's BitTorrent system. In a locality-aware variation of BitTorrent the peer is going to be served by a local peer whenever such a peer is active. Consequently, for these periods of simultaneous activity the peer is going to receive higher throughput than the one measured in our trace. We assume that the throughput offered by local peers is going to be at least as much as the maximum cumulative 5-minute throughput the user achieves throughout the trace, i.e. the peer is capable of matching its maximum upload/download rate. If we call $r(i,t)$ the rate measured in the trace for user i at time interval t, and $R(i)$ the maximum cumulative

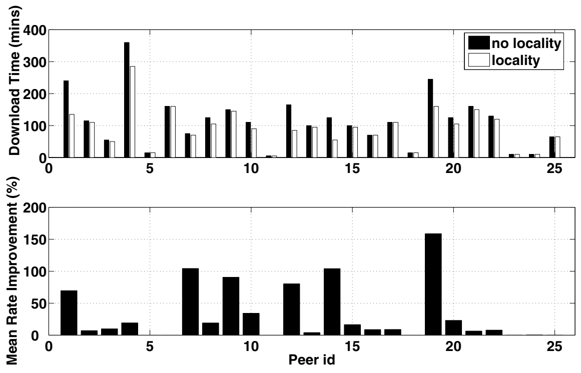


Figure 4: Implications of locality on user performance. The upper plot presents the total download time per user without and with a locality-aware p2p scheme. The lower plot presents the percent improvement in the achieved mean download rate.

rate of user i across the entire trace, then our simulation works as follows: 1) while the cumulative number of bytes downloaded by all peers is smaller than the file size, users download at their measured rates $r(i,t)$, 2) when the cumulative amount of bytes downloaded by the active users exceeds the size of the file, then all active users can download the rest of the file they need at $R(i)$ (a rate which is likely to be lower than the actual rate the user can receive within his own network), 3) when user i finishes the download of the entire file, it leaves the network - the number of active users is reduced by one and the cumulative amount of bytes in the system is reduced by the number of bytes downloaded by the departing user (e.g. the file size); this means that when the last user leaves the network the cumulative amount of bytes in the system goes back to zero. At the end of the simulation we measure the total number of bytes downloaded by each user i, and the amount of time it took for user i to finish the download. We then compute the average throughput each user i could achieve in the locality-aware case.

We ran the simulation experiment described above for the three most popular files in our packet traces. Notice that these files had at least five user requests in a day and at least one user in the Torrent downloads the file in its entirety during the one day period. In Fig. 4, we plot the average latency in the two tested scenarios (locality and no-locality) and the mean rate improvement for each user. Approximately 70% of the peers show increased mean rate in the locality scenario. The improvement ranges from minimal to more than 100% for a particular client, which experiences a mean rate improvement of up to 150%, resulting in a reduction in download time exceeding one hour. 24% of the clients experience more than 50% faster download rate, while 30% of the clients see no improvement with locality since no other user is active at the same time for that particular file and thus the file needs to be downloaded from outside the ISP. Admittedly, the sample provided by our packet traces is very small. However, given that larger networks are more likely to feature a larger pool of simultaneous active users, the associated benefits in download times are likely to be even higher.

Synopsizing our analysis of locality in our packet level traces we observe the following:

- File, byte and piece hit ratios range from 6%-18% in a day, implying that approximately one in ten files is downloaded more than once within the ISP in a day.
- Files are short-lived in the network with only a minimum number of files being requested across months.
- Active users coincide at least 30% of the time and could potentially cooperate in a locality-aware P2P system.
- At least 20% of the downloaded content existed locally in active users, while a minimum of 40% existed locally in both active and nonactive users in two of our traces.
- Peer download rates increase and download completion times per peer decrease for at least 70% of the clients in an idealized peer-assisted scenario.

4 Impact of Peer-Assisted Content Distribution on ISPs: A global perspective

In the previous section we established the benefits of locality aware peer-assisted solutions for a small ISP both in terms of ingress bandwidth as well as user experience. In this section we study the impact of peer-assisted content distribution systems on ISPs and content providers at a larger scale. We assess the overall effects of such a system, by studying the BitTorrent tracker log of the Redhat v9.0 distribution. We analyze and compare traditional client-server distribution approaches with existing P2P and caching techniques, as well as locality-aware peer-assisted mechanisms.

In the remaining of the section, we first provide a description of the tracker log and the analysis methodology, and then present our findings for the various content distribution scenarios.

4.1 Tracker log description and content distribution scenarios

The tracker log spans five months, from April to August 2003, and consists of reports conveying the progress of individual peers in downloading the 1.855GB of the Redhat v9.0 image file. As reported in [9] the user-population exhibits clear flash-crowd behavior.

Each peer sends periodic (typically every 15-30 minutes) updates to the tracker containing the amount of uploaded and downloaded data for the specific interval. Every entry of the log contains the IP address of the sender, a timestamp and the peer report to the tracker. Peer reports typically contain the following fields: *file hash*, *peer id*, *port*, *downloaded* bytes, *uploaded* bytes, *left* bytes, *event* (started/completed/stopped), *ip* (optional) and *numwant* (preferred length of the requested peer list).

Similar to our packet level analysis, identification of individual peers through the tracker messages is the first essential element in order to explore the impact of the Redhat distribution on the ISPs through the BitTorrent network. However, the procedure of distinguishing peers in the log messages poses further challenges beyond the pitfalls highlighted in section 3. First, the peer id is always random; user clients are not encoded in the peer id as, at the time, there existed one major client, namely the official BitTorrent client (http://www.bittorrent.com/). Second, user-sessions, especially multi-session downloads cannot be reconstructed in a straightforward manner (also discussed in [9]). A session is defined as a sequence of reports with the same IP and peer id which is modified with every paused and resumed session. Thus, calculating downloaded volumes per peer is problematic as the *downloaded-bytes* field refers to session and not overall bytes. Finally, users may change IPs across or within the same session.

To disambiguate individual peers we employ the following methodology:
A user is defined by the *IP and the peer id* in agreement with our practices in section 3. If the local IP exists in the peer report, we replace the IP address of the sender with the local peer IP, only if it does not correspond to private-address space (e.g., 192.168/24, etc.). The source IP address may reflect NATs or proxies and thus the local address is preferred.
To track multi-session downloads originating from the same IP as well as users behind NATs, we correlate the number of *left* bytes across consecutive reports from the same IP and varying peer ids. The left-bytes field signifies the number of bytes left to complete the overall download of the file, decreasing with time for individual peers. Thus, we regard as two separate peers, peer ids for which the number of left bytes is increasing across two consecutive reports with the same IP. On the contrary, if the number of left bytes in the current report is *less or equal* to the previous report, the two peer ids are merged into one (the most recent), potentially underestimating the user population in some cases.
We track peers changing IPs within the same /24 subnet by maintaining a mapping of all peer ids to IPs. All reports with the same peer id originating from the same /24 are treated as one peer (subject to our left-bytes condition above). We observed approximately 50K such reports (approximately 2%). Users are less likely to switch IPs outside /24 net boundaries (at least within the time-scale of minutes); however, note that users switching IPs across different ISPs (e.g., the download resumes at a different location) do not affect our analysis, and in fact they should be treated as distinct peers (downloaded/uploaded bytes affect the current ISP in each case).

After identifying the individual peers inside the network we group IP addresses into ISPs using the ASFinder module from the Coral Reef suite [12]. For the mapping we use BGP tables from RouteViews collected in May and August 2003[2]. Having a global view of where peers are located inside the network we study the impact of the following content distribution scenarios:

- A server, server farm or CDN is responsible for the content distribution (*Scenario 1*). Peers receive the content directly from the server(s) that reside outside the ISP network. Every peer request corresponds to a transfer across the ISP's Internet link.
- A distribution system based on a standard P2P system (*Scenario 2*). Here, content is distributed across peers. Peers are matched based on random selections in the network. However, we take into consideration the timing information existing in the tracker log. Peers are matched only if both of them are active at a specific time interval. Peers become active at the time of their first report in the log or with a *start* event. Similarly, we consider peers inactive after a *stop* event or in the absence of a report for the specific peer within an hour. Timing information and stop/start event reports are vital for two reasons: a) They minimize the probability of double-counting peers (smooth transition to a new peer id), and b) they allow us to reproduce the dynamics of today's popular P2P systems. If matched peers appear both within the boundaries of the same ISP, their download/upload volumes are not considered for the specific time-interval, as they do not incur any cost for the ISP (the transfer is local).
- A distribution system based on a P2P BitTorrent-like system (*Scenarios* 3&4), where peers periodically redefine their peer list based on measurements of the upload throughput offered by other peers in the network. In this case a peer obtains the requested content selecting peers randomly from groups of active users of comparable upload throughput (tit-for-tat policy). The upload bands are based on a per-IP maximum upload throughput as observed throughout the log (scenario 3), or on peer *instantaneous* upload throughputs (scenario 4). Similar to scenario 2, we consider all timing information (active/inactive peers) from the log, and transfers within the boundaries of an ISP are not taken into consideration.
- A peer-*assisted* content distribution system exploiting locality within the ISP's boundaries (*Scenario 5*). In this scenario, a peer requesting content will be redirected to any active peer (if available) within the same

[2]The use of different routing tables does not impact our findings.

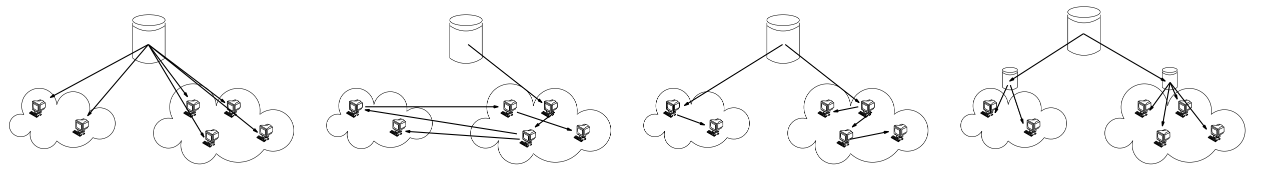

(a) Scenario 1 (b) Scenarios 2,3,4 (c) Scenario 5 (d) Scenario 6

Figure 5: Graphical representation of the examined scenarios. a) Each new download incurs additional cost for the ISP while the content provider uploads as many copies as individual users. b) The content provider uploads less copies shifting the distribution cost to the ISPs through P2P. c) An idealized peer-assisted solution ensures that one copy is downloaded per ISP while users cooperate internally. d) A caching solution where a copy is downloaded per ISP and all users are served from the local cache.

ISP (we discuss such mechanisms in Section 5). Assuming that multiple active users within an ISP have the ability to cooperate, we nominate a "leader" peer that first acquires the content and serves the rest of local users. The leader is the peer with the most available content (inferred by the left-bytes field) at every interval. Thus, when a leader is active within an ISP all other local peers are served by the leader. Note that the leader is more of a conceptual entity for byte-tracking purposes in the log, than an actual implementation of a peer-assisted solution. In reality, such a model assumes that peers have different, overlapping or not pieces of the file and cooperate by serving each other; an assumption which should not be far from reality since downloaded pieces are chosen randomly.

- A distributed caching architecture (*Scenario 6*) resembling a perfect content distribution network with caches at the access link(s) of each ISP. In this scenario, we assume infinite-space caches at the edge of the network, resulting in only one downloaded-copy of the content per ISP. Clients requesting the content are served locally by the cache. Thus, only the first download for each ISP incurs cost and will be considered. Caching would be the equivalent of a perfect peer-assisted distribution scenario, where local peers are always active and can satisfy all requests.

Fig. 5 graphically depicts the various scenarios. The content provider shifts the cost of distribution to end-nodes of the network with P2P (Fig. 5(a)&(b)), while locality mechanisms (peer-assisted Fig. 5(c) or caching Fig. 5(d)) provide savings for ingress ISP traffic.

4.2 Evaluation of content distribution scenarios

We now evaluate the cost and benefits for the content provider and the ISPs for all scenarios described in the previous section. First, we provide the evaluation metrics and a brief overview of our findings, and then present extensive results for each case.

4.2.1 Metrics and Overview

We study the benefits and costs in terms of traffic volumes for the content provider and the ISPs. All scenarios are evaluated under the same set of metrics.

To quantify the ISP cost, we measure the total ingress/egress bandwidth consumed, as well as the 95^{th} percentile of traffic in hourly intervals as observed on their access links. Such metrics are of significant interest to ISPs since they typically translate to monetary costs. Scheme performance is summarized across ISPs using the average value of the obtained distributions.

Content provider cost is measured in terms of total traffic served which corresponds to the bandwidth requirements that the provider should meet in each case. As with the ISP cost, we also measure the 95^{th} percentile of the total traffic served per hourly interval. Our findings can be summarized in the following points:

- Peer-assisted content distribution reduces ISP downlink bandwidth by a factor of two.
- A P2P locality case only requires 1.5 times the peak capacity required by a perfect caching algorithm.
- ISPs are required to upload just over only one copy of the content satisfying at the same time a large number of local peers with a local-aware solution.
- ISP savings in the peer-assisted scenario increase roughly linearly to the logarithm of active users.
- The overhead of a peer-assisted approach in terms of peak load as defined by the 95^{th} percentile, is minimal and in some cases even less than the respective overhead of a caching solution.

In the remainder of this section we expand on the above findings.

4.2.2 Impact on Downloaded Traffic

In this section, we consider the impact that peer-assisted content distribution has on the ISP's downstream traffic. A peer-assisted distribution scheme can lead to downlink bandwidth savings only if there are multiple peers concurrently downloading the same file within the ISP. If no active peers exist for the same file or the matching algorithm does not have the ability to account for them, then the benefits of peer-assisted distribution, in terms of downstream bandwidth, can be significantly reduced.

Table 3 presents the average value of the total and the 95^{th} percentile of the data downloaded by each ISP for our 6 content distribution scenarios and for both May and

August BGP tables (AS1 and AS2 respectively). P2P algorithms that match nodes at random (scenario 2) provide very little benefit in terms of ISP bandwidth savings compared to client/server distribution. In fact, a P2P algorithm with random peer matching provides less than 2% bandwidth savings over the case where the same file is distributed once for each client.

However, current P2P systems such as BitTorrent do not rely on a completely random matching of nodes. As discussed in previous sections, BitTorrent features an algorithm that leads to peers clustering according to their upload capacities. Such a P2P algorithm may result in locality, if nodes within an ISP were to have similar download/upload speeds.

Table 3 shows that BitTorrent-like systems improve the locality as more local clients are matched, however, the extra benefits compared to a completely random selection are almost negligible. A potential reason is that the user population for each throughput band is quite large, ergo random peer selection within the band is more likely to lead to a peer outside the ISP.

On the contrary, peer-assisted locality solutions offer high potential benefits. Our results show that *a peer-assisted locality-aware scheme can reduce the ISP's ingress link utilization by a factor of two*.

Finally, the perfect caching solution provides the best possible benefits for the ISP since only one copy of the file is downloaded regardless of the number of internal requests. Compared to the caching infrastructure, the peer-assisted locality-aware solution results in several times the optimal bandwidth requirements, since peers are not always active resulting in multiple downloads per ISP. Nevertheless, one should note that in absolute terms, the amount of traffic generated by such a peer-assisted scheme is only a small fraction of what a client/server solution would produce.

In terms of the 95^{th} percentile, Table 3 demonstrates that locality-aware solutions perform much closer to a caching infrastructure. In most cases, *a P2P solution requires only approximately* 1.5 *times the amount of peak capacity required by a caching solution*. P2P systems are most helpful when demand is high since a large number of simultaneous users provides increasing opportunities for cooperation. However, when user populations is low, P2P systems like BitTorrent almost revert to a client/server model. Thus, P2P systems even though they do not operate at the optimal when decreasing the total amount of traffic, they appear most beneficial when they are needed the most (e.g. during peak loads).

4.2.3 Impact on Uploaded Traffic

Since peers become servers in the P2P paradigm, ISPs are bound to observe increasing amounts of egress traffic, which may considerably impact their bandwidth cost.

Table 3: Downloaded data (in MB) by each ISP. Percentages show savings compared to the client/server model.

	AS1 (Avrg)	AS2 (Avrg)	AS1 (95^{th})	AS2 (95^{th})
Sc.1	14137	15313	804	862
Sc.2	13954 (1.3%)	15110 (1.3%)	794 (1.3%)	854 (1%)
Sc.3	13784 (2.5%)	14920 (2.6%)	786 (2.2%)	843 (2.3%)
Sc.4	13872 (1.9%)	15023 (1.9%)	791 (1.7%)	852 (1.1%)
Sc.5	6710 (52.5%)	7243 (52.7%)	625 (22.3%)	688 (20.2%)
Sc.6	1191 (91.6%)	1268 (91.7%)	459 (42.9%)	490 (43.2%)

Table 4: Average uploaded data (in MB) by each ISP.

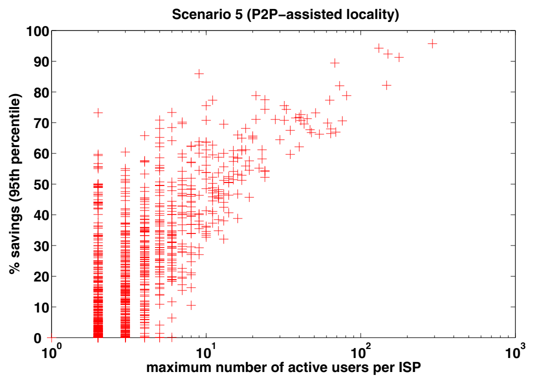
	AS1 (avg)	AS2 (avg)	AS1 (95^{th})	AS2 (95^{th})
Client/Server	-	-	-	-
P2P Random	17239	18188	750	789
P2P BitT	17551	18538	759	808
P2P Locality savings	2827 84%	2971 84%	238 68%	248 69%
Caching	-	-	-	-

This imposed cost is evident in table 4, where simple P2P content distribution results in high upstream bandwidth requirements compared to the traditional client server model (or the caching infrastructure) where local users do not serve content.

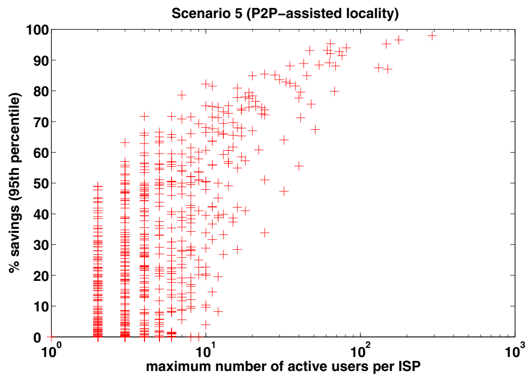
On the contrary, locality keeps most of the traffic within the ISP's boundaries while the amount of traffic uploaded externally is reduced by more than a factor of 6. *In fact most ISPs only need to upload slightly over one copy of the file in order to satisfy a large number of internal users*. Examination of the 95^{th} percentile offers similar observations when comparing current P2P with locality aware systems. However, cross-examination of the peak capacity across tables 3 and 4 shows that the peak upload capacity required is much smaller than the download capacity.

4.2.4 Impact vs. ISP size

We now examine how peer-assisted content distribution affects ISPs depending on their size. Fig. 6 shows the savings in terms of total traffic and the 95^{th} percentile offered by a perfect peer-assisted locality-aware solution compared to a client-server solution depending on the maximum number of active users inside an ISP (which is a rough indication of the ISP's size). Fig. 6(a) shows that savings from the peer-assisted locality-aware solution increase almost linearly with the logarithm of the number of active users. While small ISPs do not experience high benefits as expected, medium and large-size ISPs greatly benefit from the peer-assisted locality-aware system. *In fact, the benefits for ISPs with more than thirty maximum active users are higher than* 60%. In terms of the 95^{th} percentile, the benefits are even higher for the same population size; for a system with a maximum number of active users equal to thirty, the benefits are approximately 80% (Fig. 6(b)).

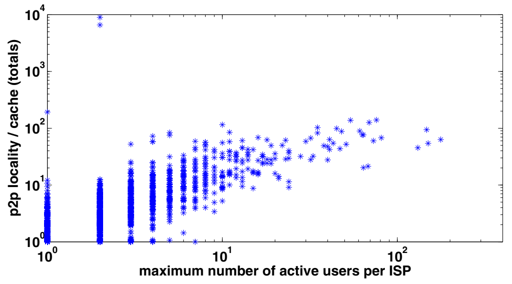


(a) Savings vs. maximum number of active users

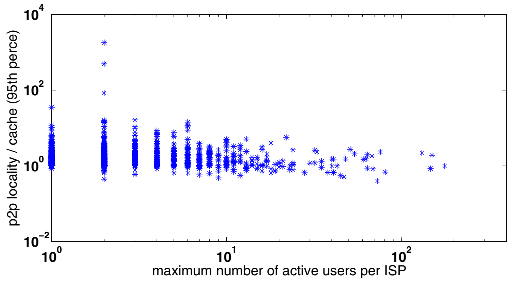


(b) 95th percentile of savings vs. maximum number of active users

Figure 6: Savings of a peer-assisted distribution versus the traditional client/server model in terms of total traffic and the 95^{th} percentile with respect to the ISP size (maximum number of active users). ISPs with more than thirty active users experience savings of 60% and 80% for total and peak traffic respectively.



(a) Overhead of a locality-aware algorithm over a perfect caching system as a function of the ISP's size



(b) Overhead of a locality-aware algorithm over a perfect caching system as a function of the ISP's size (95th percentile)

Figure 7: Comparison of the peer-assisted distribution versus a perfect caching infrastructure. While in terms of total traffic the overhead appears high, the two scenarios appear equivalent in terms of peak traffic. The overhead in terms of total traffic flattens out with increasing ISP size.

4.2.5 P2P Locality-Aware vs Proxy Caches

We now examine how optimal the peer-assisted solution appears compared to per-ISP caching.

Figure 7 presents the ratio of bytes downloaded by a peer-assisted solution versus a caching solution. The overhead of a peer-assisted solution can be quite high compared to the caching solution for small ISPs, since the request rate is low. As the ISP size increases, the overhead of the peer-assisted solution also increases and large ISPs need to download a larger number of copies compared to a caching solution. However, the overhead appears to flatten out at large ISP sizes since large number of active users ensures cooperation.

On the contrary the overhead of a a locality-aware solution is small compared to caching in terms of the 95^{th} percentile. Indeed, *the* 95^{th} *percentile of the peer-assisted solution is equal to or even lower from the one of the caching scenario*. The lower 95^{th} percentile in the peer-assisted case results from the fact that most peak rates observed in the peer-assisted solution are lower than the peak rate of the caching scheme, thus pushing the percentile closer to the mean rate.

Table 5: Total egress server capacity

	AS1 (avg)	AS2 (avg)	AS1 (95^{th})	AS2 (95^{th})
Client-Server	59.8 TB	59.8 TB	17 TB	17 TB
P2P Locality savings	28.4 TB 52.5%	28.3 TB 52.7%	8.1 TB 52.3%	8.1 TB 52.3%
Caching savings	5 TB 91.6%	5 TB 91.7%	1.6 TB 91%	1.6 TB 91%

4.2.6 Impact of Locality on the Content Provider

Finally, we consider the reduction in bandwidth enjoyed by the content provider when a peer-assisted locality-aware solution is used. To estimate the amount of data served by the content provider in the peer-assisted case, we assume that requests which are not satisfied within an ISP need to be satisfied by the content provider. This is an upper bound of the amount of bandwidth required from the content provider since a fraction of the requests would also be satisfied by other ISPs in a P2P system.

Table 5 demonstrates that *locality results in less than half the resource requirements when compared to the client-server model both in terms of total egress traffic and* 95^{th} *percentile*. Further, a caching solution reduces the total egress capacity by one order of magnitude.

Table 6: Downloaded data (in MB) by each ISP for different locality algorithms.

	/24	/16	DOMAIN	Hierarchical	Proxy Tracker
P2P Locality (Avrg)	13964 (1.2%)	11643 (17.7%)	10864 (23.1%)	10227 (27.5%)	6710 (52.5%)
P2P Locality (95^{th})	779 (3.1%)	698 (13.2%)	709 (11.8%)	689 (14.3%)	625 (22.3%)

Table 7: Uploaded data (in MB) for each ISP.

	/24	/16	DOMAIN	Proxy Tracker
P2P Loc. (Avrg)	5415	4656	4735	2827
P2P Loc. (95^{th})	317	286	289	238

5 Locality Algorithms and their Performance

In this section we describe two simple potential implementations of a peer-assisted content distribution mechanism and compare their performance. Locality may be implemented either by ISPs or be imposed by the content provider. ISPs may implement a locality-aware BitTorrent-like system by deploying *proxy-trackers* at the edges of their network. Proxy-trackers will then intercept requests of local peers and redirect them to any existing active users within the boundaries of the ISP. This practice corresponds to our peer-assisted locality scenario which we extensively studied throughout the previous section. However, such infrastructure-based solutions are not always possible to deploy and maintain.

Alternative locality-based algorithms may be supported by the content provider without relying on extra infrastructure being deployed. Such locality solutions could be implemented with small modifications to BitTorrent trackers and could consider algorithms based on simple prefix rules, domain names, or more sophisticated hierarchical prefix matching rules, network-aware clustering [13] or routing table lookups (the two latter cases would lead to performance very similar to the proxy-tracker scenario).

To evaluate the performance of such algorithms, we assume a certain division of clients among ISPs based on AS information from our May BGP table (results are similar for August). Then, we compare the performance of different locality algorithms versus a perfect P2P locality-aware system that would utilize a proxy-tracker at the edge of each ISP. Note however that the resulting matching of peers in the peer-assisted scheme will be inefficient when the imposed locality algorithm does not match AS boundaries (for example a locality algorithm based on a /16 prefix matching).

Tables 7 and 6 show the amount of data downloaded and uploaded by each ISP depending on the locality algorithm used. The percentage results in brackets are the savings with respect to a client/server solution. Table 6 shows that locality solutions that try to match clients within the same DNS domain are not as efficient as proxy-tracker solutions, however, they provide roughly 50% of the overall benefits. The limitations of such a solution arise from the fact that different clients in the same domain may span multiple ASes, thus, generating large amount of cross-over traffic among ISPs.

Static prefix-based solutions (e.g. /24, /16) that match users within a certain prefix perform overall slightly worse than a DNS based solution. Small prefixes (e.g. /24) result in creating small groups with a limited population of active users, while on the contrary, very large prefixes result in users being too spread across multiple ISPs. Our results indicate that the best prefix-based grouping is /13, which performs slightly better than the domain-matching case.

Finally, we also examine the performance of a hierarchical-prefix matching solution where users are first matched within a /24 prefix, then, /16, /14, and finally /13. The advantages of such a hierarchical solution would be that a) peer-matching can dynamically accommodate ISPs of different sizes, and b) nearby peers within an ISP are matched first. The hierarchical-matching algorithm provides the best performance after a proxy-tracker matching scheme, although, it still fails to match a large number of local peers. This may be due to the fact that ASes typically own non-consecutive ranges of IP addresses, a practice that decreases the effectiveness of any prefix-matching algorithm.

6 Related Work

Previous work on BitTorrent has focused on measurements [4, 9], theoretical analysis [22], and improvements [27]. Izal et al. analyze the log of a BitTorrent tracker showing the flash-crowd effect of a single file, download speeds, and the amount of time that peers stay after they have completed the download [9]. Pouwelse et al. present an extensive analysis of BitTorrent showing availability, peer uptimes, and providing a better understanding of peer interarrival times [21].

Apart from BitTorrent, several measurement studies have addressed the issues of availability [2, 3, 8], integrity [29], flash-crowds [4][14], and download performance [1][26][25][3] in other P2P systems. Saroiu et al. use SProbe (sprobe.cs.washington.edu) to measure the bandwidth of Gnutella peers [25]. Liang et al. provide an extensive study of the performance of the KaZaA network [15]. An analysis of Gnutella traces in terms of resource demand, popularity of particular search terms, overlay topology, and node latency was presented by Nogueira et al. [18]. Gnutella data, was also examined by Ripeanu and Foster [23], focusing on node connectivity, overlay topology, and protocol message overhead. A trace analysis of the network traffic from the perspective of traffic characterization and bandwidth provisioning was presented by Sen and Wang [26]. Markatos [16] conducted a study of Gnutella protocol traffic aimed at caching query/response traffic to reduce network load. Leibowitz et al [14] examined Kazaa traffic to determine proportion of total network traffic by file popularity.

To the best of our knowledge, this paper presents the first rigorous study of the impact that peer-assisted content distribution solutions have on ISPs from both a local and a global perspective. The fact that users in peer-assisted solutions only form a sharing community only while downloading the same file, significantly differentiates them from other existing file-sharing applications and has important implications in the potential benefit of locality-based solutions. For instance, [24] provides an extensive analysis of content delivery systems, including CDN caching, KaZaa, and Gnutella. However, their results do not carry over well to peer-assisted solutions such as BitTorrent where cooperation only happens if clients are active and sharing the same file.

Regarding the locality analysis, previous studies have proposed new ways of clustering peers (e.g. [5][19][20]) and studied the potential benefits of locality in P2P file-sharing systems such as KaZaa and Gnutella [8][24]. In this work, we quantify the potential benefits of peer-assisted locality-aware solutions in a real setting and study the benefits of very simple locality algorithms that require minor changes to existing solutions. Moreover, we study the potential benefit that locality-based solutions have for the content provider and the clients in terms of bandwidth reduction and decreased download times respectively.

7 Discussion

Our analysis thus far has presented a detailed description of the cost and benefits of locality-aware peer-assisted content distribution. However, we would like to stress here a number of implications based on our findings.

Global vs. local benefits: Our analysis presents two different perspectives of the potential long-term benefits of peer-assisted content distribution. While in the global case total savings appear significant, locality appears to only offer minimal savings in our monitored network. This discrepancy is due to various factors, some inherent to each case, others reflecting our measurement data. First, since savings depend on the ISP size (section 4), benefits are limited for our monitored network (smaller than typical ISPs with a limited number of simultaneous active users). Further, while the tracker log covers a period of six months (including the initial flash crowd effect), the duration of all our packet traces is roughly a day thus hiding potential benefits. Finally, locality analysis for our monitored ISP reflects a large number of files (unpopular files or files past the flash crowd effect reduce the observed hit ratios) versus one tracker in the Redhat log data.

Peer-assisted vs. existing content distribution solutions: Our analysis does not in any way suggest that peer-assisted solutions should replace current content distribution practices such as CDNs. Issues such as file availability with small user population, limited end-to-end connectivity with NATs, security, reliability and service guarantees need to be robustly resolved before such solutions can be deployed in a commercial setting. In the meantime, our findings suggest that peer-assisted schemes will definitely prove beneficial when complementing current practices.

Impact of peer-assisted content distribution on internal ISP traffic: While locality-aware peer-assisted solutions appear attractive for ISPs, we would like to stress here that decreasing egress traffic may create the need for traffic re-engineering within ISP boundaries to account for the additional internal upload traffic. Increased upload traffic may prove especially important for those ISPs that depend on other carriers to deliver last-mile traffic (e.g., European Tier-2 ISPs). In fact, for such ISPs, utilizing their transit egress capacity might result in a more cost-effective solution rather than re-routing traffic internally. Caching, on the contrary, fits naturally with the traditional asymmetric architecture of today's ISPs, where the downstream channel is more heavily provisioned relative to the upstream, driven by the assumption that customers download more than what they upload.

8 Conclusions

Based on payload packet traces as well as tracker-based logs, we have studied the impact that local-aware peer-assisted content distribution solutions can have on ISPs, content providers, and end-users. In particular, we have identified that current P2P solutions are very ISP-unfriendly, generating large amount of unnecessary traffic both downstream as well as upstream.

We studied locality in the context of BitTorrent. Our traces indicate that BitTorrent is locality-unaware, severely increasing ISPs' bandwidth requirements. In particular, up to 70-90% of *existing local* content was found to be downloaded from external peers.

In BitTorrent, users typically only share content while their download is active. Our results show that such a feature does not significantly impact the benefits of a BitTorrent-like solution. In fact, we found that users requesting the same file within an ISP overlap 30%-70% of the time and could, therefore, efficiently help each other if a locality algorithm were in place. Furthermore, by having users stay longer after the download is complete and share their content, the potential benefit of a locality algorithm would be an extra 20%-40% in terms of reduced downloaded bytes by the ISP.

Peer-assisted content distribution incurs significant upstream capacity costs for the transit links (roughly doubling the bandwidth requirements). However, simple locality based mechanisms can rectify this effect, approximating the performance of a perfect caching architecture. Overall, locality-aware peer-assisted algorithms decrease the bandwidth of the content provider's egress link by more than a factor of two. Strategies such as those used by BitTorrent trying to match users with similar capacities provide little locality benefits.

The benefits of a peer-assisted locality solution increase with the logarithm of the number of active users. Our findings show that as soon as there are more than 30 active users within an ISP, a peer-assisted locality solution provides more than 60% savings in terms of ISP's ingress traffic compared to a client/server distribution.

On the contrary, a peer-assisted locality-aware solution generates five times more traffic on average through the ISP's link than a perfect caching solution. However, in absolute terms this represents only a small fraction of the traffic generated by a client/server solution.

The benefits of a peer-assisted solution are always much more pronounced in terms of 95^{th} percentile, thus, absorbing peak loads and reducing the monetary impact on ISPs and content providers. Simple locality-aware mechanisms based on domain-name grouping, or prefix grouping provide roughly 50% of the potential benefits.

Our study shows that while current peer-assisted content distribution solutions are ISP-unfriendly, this is not a fundamental limitation and that minor modifications can indeed significantly reduce the costs of all parties involved in the content distribution process. Such simple modifications to peer-assisted protocols can provide a cost-effective solution that can be exploited by content providers to scale and accelerate the delivery of content to millions of users without pushing ISPs towards regulating or blocking such traffic.

Acknowledgments

The authors are thankful to Ernst W. Biersack for providing the RedHat tracker log and to Andrew Moore for facilitating this study.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[2] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *International Workshop on Peer to Peer Systems*, 2003.

[3] J. Chu, K. Labonte, , and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *ITCom: Scalability and Traffic Control in IP Networks*, 2002.

[4] B. Cohen. Incentives build robustness in Bittorrent. . In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[5] M. Costa, M. Castro, A. Rowstron, , and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *ICDCS*, 2004.

[6] Ethereal. http://www.ethereal.com/.

[7] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *IEEE/INFOCOM*, 2005.

[8] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP*, 2003.

[9] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *PAM*, 2004.

[10] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *WWW*, 2002.

[11] T. Karagiannis, A.Broido, M. Faloutsos, and kc claffy. Transport layer identification of P2P traffic. In *ACM Sigcomm IMC*, 2004.

[12] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k. claffy. The architecture of the CoralReef: Internet Traffic monitoring software suite. In *PAM*, 2001.

[13] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *SIGCOMM*, 2000.

[14] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network. In *3rd IEEE Workshop on Internet Applications (WIAPP'03)*, 2003.

[15] J. Liang, R. Kumar, and K. W. Ross. The KaZaA Overlay: A Measurement Study. In *Computer Networks (Special Issue on Overlays)*, to appear.

[16] E. Markatos. Tracing a large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[17] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt. Architecture of a Network Monitor. In *PAM*, 2003.

[18] D. Nogueira, L. Rocha, J. Santos, P. Araujo, V. Almeida, and W. Meira. A methodology for workload characterization of file-sharing peer-to-peer networks. In *IEEE Workshop of Workload Characterization*, 2004.

[19] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *ACM Sigcomm*, 2001.

[20] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *IPTPS*, 2003.

[21] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *IPTPS*, 2005.

[22] D. Qiu and R. Srikant. Modeling and performance analysis of bit torrent-like peer-to-peer networks. In *ACM Sigcomm*, 2004.

[23] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *IPTPS*, 2002.

[24] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems . In *OSDI*, 2002.

[25] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *MMCN*, 2002.

[26] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. In *ACM SIGCOMM IMW*, 2002.

[27] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. In *Infocom*, 2004.

[28] Bittorrent Protocol Specification v1.0. http://wiki.theory.org/BitTorrentSpecification.

[29] H. Weatherspoon and J. Kubiatowicz. Naming and Integrity: Self-Verifying Data in Peer-to-Peer Systems. In *FuDiCo*, 2002.

Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations

Qi (George) Zhao Abhishek Kumar Jun (Jim) Xu
College of Computing, Georgia Institute of Technology

Abstract

Detecting the sources or destinations that have communicated with a large number of distinct destinations or sources during a small time interval is an important problem in network measurement and security. Previous detection approaches are not able to deliver the desired accuracy at high link speeds (10 to 40 Gbps). In this work, we propose two novel algorithms that provide accurate and efficient solutions to this problem. Their designs are based on the insight that sampling and data streaming are often suitable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is an excellent way to recover the complete information. Our first solution builds on the standard hash-based flow sampling algorithm. Its main innovation is that the sampled traffic is further filtered by a data streaming module which allows for much higher sampling rate and hence much higher accuracy. Our second solution is more sophisticated but offers higher accuracy. It combines the power of data streaming in efficiently estimating quantities associated with a given identity, and the power of sampling in collecting a list of candidate identities. The performance of both solutions are evaluated using both mathematical analysis and trace-driven experiments on real-world Internet traffic.

1 Introduction

Measurement of flow-level statistics, such as total active flow count, sizes and identities of large flows, per-flow traffic, and flow size distribution are essential for network management and security. Measuring such information on high-speed links (e.g., 10 Gbps) is challenging since the standard method of maintaining per-flow state (e.g., using a hash table) for tracking various flow statistics is prohibitively expensive. More specifically, at very high link speeds, updates to the per-flow state for each and every incoming packet would be feasible only through the use of very fast and expensive memory (typically SRAM), while the size of such state is very large [7] and hence too expensive to be held in SRAM. Recently, the techniques for approximately measuring such statistics using a much smaller state, based on a general methodology called *network data streaming*, have been used to solve some of the aforementioned problems [5, 6, 12, 11, 22]. The main idea in network data streaming is to use a small and fast memory to process each and every incoming packet in real-time. Since it is impractical to store all information in this small memory, the principle of data streaming is to maintain only the information most pertinent to the statistic to be measured.

In this work, we design data streaming algorithms that help detect *super sources and destinations*. A super source[1] is a source that has a large *fan-out* (e.g., larger than a predefined threshold) defined as the number of distinct destinations it communicates with during a small time interval. The concepts of super destination and *fan-in* can be defined symmetrically. Our schemes in fact solve a strictly harder problem than making a binary decision of whether a source/destination is a super source/destination or not: They actually provide accurate estimates of the fan-outs/fan-ins of potential super sources/destinations. In this work a *source* can be any combination of "source" fields from a packet header such as source IP address, source port number, or their combination, depending on target applications. Similarly, a *destination* can be any combination of the "destination" fields from a packet header. We refer to the *source-destination pair* of a packet as the *flow label* and use these two terms interchangeably in the rest of this paper.

The problem of detecting super sources and destinations arises in many applications of network monitoring and security. For example, port-scans probe for the existence of vulnerable services across the Internet by trying to connect to many different pairs of destination IP address and port number. This is clearly a type of super source under our definition. Similarly, in a DDoS (Distributed Denial of Service) attack, a large number of zombie hosts flood packets to a destination. Thus the problem of detecting the launch of DDoS attacks can be viewed as detecting a super destination. This problem also arises in detecting worm propagation and estimating their spreading rates. An infected host often propagates the worm to a large number of destinations, and can be viewed as a super source. Knowing its fan-out allows us to estimate the rate at which the worm may spread. Another possible instance lies in peer-to-peer and content distribution networks, where a few servers or peers might attract a larger number of requests (for content) than they can handle while most of others in the network are relatively idle. Being able to detect such "hot spots" (a type of super destination) in real-time helps balance the workload and improve the overall performance of the network. A number of other variations of the above applications, such as detecting flash crowds [9] and reflector attacks [15], also motivate this problem.

Techniques proposed in the literature for solving this problem typically maintain per-flow state, and cannot scale to high link speeds of 10 or 40 Gbps. For example, to detect port-scans, the widely deployed Intrusion Detection System (IDS) *Snort* [19] maintains a hash table of the distinct source-destination pairs to count the destinations each source talks to. A similar technique is used in FlowScan [17] for detecting DDoS attacks. The inefficiency in such an approach stems from the fact that most of the source-destination pairs are not a part of port scans or DDoS attacks. Yet, they result in a large number of source-destination pairs that can be accommodated only in DRAM, which cannot support the high access rates required for updates at line speed. More recent work [20] has offered solutions based on hash-based flow sampling technique. However, its accuracy is limited due to the typically low sampling rate imposed by some inherent limitations of the hash-based flow sampling technique discussed later in Section 3. A more comprehensive survey of related work is provided in Section 7.

In this paper we propose two efficient and accurate data streaming algorithms for detecting the set of super sources by estimating the fan-outs of the collected sources. These algorithms can be easily adapted symmetrically for detecting the super destinations. Their designs are based on the insight that (flow) sampling and data streaming are often suitable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is an excellent way to recover the complete information. This insight leads to two novel methodologies of combing the power of streaming and sampling, namely, "filtering after sampling" and "separation of counting and identity gathering". Our two solutions are built upon these two methodologies respectively.

Our first solution, referred to as the *simple scheme*, is based on the methodology of "filtering after sampling". It enhances the traditional hash-based flow sampling algorithm to approximately count the fan-outs of the sampled sources. As suggested by its name, the design of this solution is very simple. Its main innovation is that the sampled traffic is further filtered by a simple data streaming module (a bit array), which guarantees that at most one packet from each flow is processed. This allows for much higher sampling rate (hence much higher accuracy) than achievable with traditional hash-based flow sampling. Our second solution, referred to as the *advanced scheme*, is more sophisticated than the simple scheme but offers even higher accuracy. Its design is based on the methodology of "separation of counting and identity gathering", which combines the power of streaming in efficiently estimating quantities (e.g., fan-out) associated with a given identity, and the power of sampling in generating a list of candidate identities (e.g., sources). Through rigorous theoretical analysis and extensive trace-driven experiments on real-world Internet traffic, we demonstrate these two algorithms produce very accurate fan-out estimations.

We also extend our advanced scheme for detecting the sources that have large *outstanding fan-outs*, defined as the number of distinct destinations it has contacted but has not obtained acknowledgments (TCP ACK) from. This extension has several important applications. One example is that in port-scans, the probing packets, which target a large number of destinations, will receive acknowledgments from only a small percentage of them. Another example is distributed TCP SYN attacks. In this case, the victim's TCP acknowledgments (SYN/ACK packets) to a large number of hosts for completing the TCP handshake (the second step) are not acknowledged. Our evaluation on bidirectional traffic collected simultaneously on a link shows that our solution estimates outstanding fanout with high accuracy.

The rest of this paper is organized as follows. In the next section, we present our design methodologies and provide an overview of the proposed solutions. Sections 3 and 4 describe the design of the two schemes in detail respectively and provide a theoretical analysis of their complexity and accuracy. Section 5 presents an extension of our scheme for estimating outstanding fan-outs. We evaluate our solutions in Section 6 using packet header traces of real-world Internet traffic. We discuss the related work in Section 7 before concluding in Section 8.

2 Overview of our schemes

As we mentioned above, accurate measurement and monitoring in high speed networks are challenging because the traditional per-flow schemes cannot scale to high link speeds. As a stop-gap solution, packet sampling has been used to keep up with high link speeds. In packet sampling, a small fraction p of traffic is sampled and processed. Since the sampled packets constitute a much lower volume than the original traffic, a per-flow table stored in relatively inexpensive DRAM can handle all the updates triggered by the sampled packets in real-time [14]. Thus we can typically obtain complete information contained in the sampled traffic. The statistics of the original traffic are then inferred from that of the sampled traffic by "inverting" the sampling process, i.e., by compensating for the effects of sampling. However the accuracy of such sampling-based estimations is usually low, because the error is scaled by $1/p$ and p is typically small (e.g., 1/500) to make the sampling operation computationally affordable [4, 11, 8]. In other words, although the sampling-based approach allows for 100% accurate digesting of information on sampled traffic, a large amount of information may be lost during the sampling process.

Network data streaming[2] has begun to be recognized as a better alternative to sampling for measurement and monitoring of high-speed links [10]. Contrary to sampling, a network data streaming algorithm will process *each and every packet* passing through a high-speed link to glean the most important information for answering a specific type of query, using a small yet well-organized data structure. This data structure is small enough to be fit into fast (yet expensive) SRAM, allowing it to keep up with high link speeds. The challenge is to design this data structure in such a way that it encodes the information we need, for answering the query, in a succinct manner. Data streaming algorithms, if available, typically offer much more accurate estimations than sampling for measuring network flow statistics. This is because, intuitively the sampling throws away a large percentage of information up front, while data streaming, which processes each and every packet, is often able to retain most of the most important information inside a small SRAM module.

In our context of detecting super sources, however, both sampling and data streaming are valuable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is used to recover the complete information. There are two parts of information that we would like to know in finding super sources: one is the identities (e.g., IP addresses) of the sources that may be super sources. The other is the fan-out associated with each source identity. We observe that data streaming algorithms can encode the fan-outs of various sources into a very succinct data structure. Such a data structure, however, typically only provides a lookup interface. In other words, if we obtain a source identity through other means, we are able to look up the data structure to obtain its (approximate) fan-out, but the data structure itself cannot produce any identities and is undecodable without such identities being supplied externally. On the other hand, sampling is an excellent way of gathering source identities though it is not a great counting device as we described earlier.

The above observation leads to one of the two aforementioned design methodologies, i.e., separating identity gathering and counting. The idea is to use a streaming data structure as a counting device and use sampling to gather the identities of potential super sources. Then we look up the streaming data structure using the gathered identities to obtain the corresponding counts. This methodology is used in our advanced scheme that employs a 2-dimensional bit array as the counting device, in parallel with an identity gathering module that adopts an enhanced form of sampling. We show that our sampling module has vanishingly small probability of missing the identity of any actual super sources and the estimation module produces highly accurate estimates of the fan-out of the potential super sources. This scheme is especially suitable for very high link speeds of 10 Gbps and above. We describe this scheme in Section 4.

We also explore another way of combing sampling and streaming, i.e., "filtering after sampling". Its idea is to employ a data streaming module between the sampling operation and the final processing procedure to efficiently encode whether a flow has been seen before. A careful design of this module guarantees that at most one packet in each flow needs to be processed. This allows us to achieve much higher sampling rate and hence much higher accuracy than the traditional flow sampling scheme. This solution works very well for relatively lower link speeds (e.g., 10 Gbps and below). We describe this scheme in detail in Section 3.

3 The simple scheme

In this section we present a relatively simple scheme for detecting super sources. It builds upon the traditional hash-based flow sampling technique but can achieve a much higher sampling rate, and hence more accurate estimation. We begin with a discussion of some limitations of the traditional hash-based sampling approach, and then describe our solution that alleviates these limitations. We also present an analysis of the complexity and accuracy of the scheme.

3.1 Limitations of traditional hash-based flow sampling

There are two generic sampling approaches for network measurement: packet sampling and flow sampling. In the former approach, each packet is sampled independently with a certain probability, while in the latter, the sampling decision is made at the granularity of flows (i.e., all packets belonging to sampled flows are sampled). In the following, we only consider flow sampling since packet sampling is not suitable for our context of detecting super sources. [3]

A traditional flow sampling algorithm that estimates the fan-outs of sources works as follows. The algorithm randomly samples a certain percentage (say p) of source-destination pairs using a hashing technique (described next). The fan-out of each source in the sampled pairs is counted and then scaled by $1/p$ to obtain an estimate of the fan-out of the source in the original traffic (i.e., before sampling). This counting process is typically performed using a hash table that stores the fan-out values (after sampling) of all sources seen in the sampled traffic so far, and a newly sampled flow will increment the fan-out counter of the corresponding hash node (or trigger the creation of a new node). Since the estimation error is also scaled by $1/p$, it is desirable to make the sampling rate p as high as possible. However, we will show that, at high link speeds, the traditional hash-based flow sampling approach may prevent us from achieving high sampling rate needed for accurate estimation.

Flow sampling is commonly implemented using a simple hashing technique [3] as follows. First a hash function g that maps a flow label to a value uniformly distributed in $[0, 1)$ is fixed. When a packet arrives, its flow label is hashed by g. Given a sampling probability p, the flow is sampled if and only if the hashing result is no more than p. Recall that the purpose of flow sampling is to reduce the amount of traffic that needs to be processed by the aforementioned hash table which performs the counting. Clearly, it is desirable that a hash table that runs slightly faster than p times the link speed, can keep up with the incoming rate of the sampled traffic (with rate p). For example, we would like a hash table (in DRAM) that is able to process a packet in $400ns$ to handle all traffic sampled from a link with 10 million packets per second (i.e., one packet arrival per $100ns$ on the average) with slightly less than 25% sampling rate. Unfortunately, we cannot achieve this goal with the current hash-based flow sampling approach for the following reason.

With hash-based flow sampling, if a flow is sampled, all packets belonging to the flow need to be processed by the hash table. Internet traffic is very bursty in the sense that the packets belonging to a flow tend to arrive in bursts and do not interleave well with packets from other flows and is also known to exhibit the following characteristic [5]: a small number of elephant flows contain most of the overall traffic while the vast majority of the flows are small. If a few elephant flows are sampled, their packets could generate a long burst of sampled traffic that has much higher rate than that can be handled by the hash table[4]. Therefore, with hash-based flow sampling, the sampling rate p has to be much smaller than the ratio between the operating speed of the hash table and the arrival rate of traffic, thus leading to large estimation errors as discussed before. In the following subsection, we present an efficient yet simple solution to this problem, allowing the sampling rate to reach or even well exceed this ratio.

In [20] the authors propose a *one-level filtering algorithm* which uses the hash-based flow sampling approach described above, in conjunction with a hash table for counting the fan-out values. It does not specify whether DRAM or SRAM will be used to implement the hash table. If DRAM were used, it will not be able to achieve a high sampling rate as discussed before. If SRAM were used, the memory cost is expected to be prohibitive when the sampling rate is high. This algorithm appears to be effective and accurate for monitoring lower link speeds, but cannot deliver a high estimation accuracy when operating at high link speeds such as 10Gbps (the target link speeds are not mentioned in [20]).

3.2 Our scheme

We design a filtering technique that completely solves the aforementioned problem. It allows the sampling rate to be very close to the ratio between the hash table speed and the link speed in the worst-case and well exceed the ratio otherwise. Its conceptual design is shown in Figure 1. Compared with the traditional flow sampling approach, our approach places a data streaming module between the hash-based flow sampling module and the hash table (for counting). *This streaming module guarantees that at most one packet from each sampled flow needs to be processed by the hash table.* This will completely smooth out the aforementioned traffic bursts in the flow-sampled traffic, since such bursts are caused by highly bursty arrivals from one or a small number of elephant flows and now only the first packets of these flows may trigger updates to the hash table.

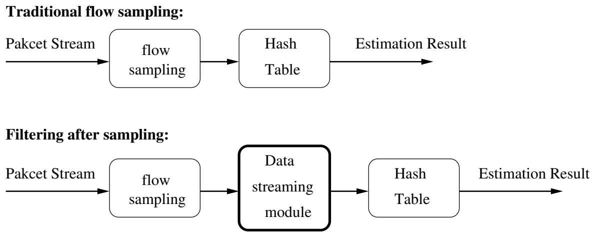


Figure 1: Traditional flow sampling vs. filtering after sampling

```
1.  Initialize
2.    $G[r] := 0$, r=1,2,..., $w$
3.    /* $w$ is the size of the array */
4.    $u := w$
5.    /* variable $u$ keep track the number of "0"s in $G$ */

6.  Filtering after sampling
7.    Upon each incoming sampled packet $pkt$
8.      $r := h(< pkt.src, pkt.dst >)$
9.      if $G[r] = 0$
10.       $s := pkt.src$
11.       $\widehat{N_s} := \widehat{N_s} + \frac{w}{u}$
12.       /* The ($s$, $\widehat{N_s}$) pairs are maintained as a hash
            table $L$. */
13.       $G[r] := 1$
14.       $u := u - 1$
15.       /* The number of "0"s is decreased by 1 */
```

Figure 2: Algorithm of updating data streaming module.

The data structure and algorithm of the data streaming module are shown in Figure 2. Its basic idea is to use a bit array G to remember whether a flow label, a source-destination pair in our context, has been processed by the hash table. Let the size of the array be w bits. We fix a hash function h that maps a flow label to a value uniformly distributed in $[1, w]$. The array is initialized to all "0"s at the beginning of a measurement epoch. Upon the arrival of a packet pkt, we hash its flow label ($< pkt.src, pkt.dst >$) using h and the result r is treated as an index into the array

G. If $G[r]$ is equal to 1, our algorithm concludes that a packet with this flow label has been processed earlier, and takes no further action. Otherwise (i.e., $G[r]$ is 0), this flow label will be processed to update the corresponding counter $N_{pkt.src}$ maintained in a hash table L. Then $G[r]$ is set to 1 to remember the fact that a packet with this flow has been seen and processed. This method clearly ensures that at most one packet from each sampled flow is processed by L. However, due to hash collisions, some sampled flows may not be processed at all since their corresponding bits in G would be set by their colliding counterparts.[5] The update procedure of the hash table L, described next, statistically compensates for such collisions.

Now we explain our statistical estimator, which is the computation result of the hash table update procedure shown in Figure 2 (line 11). Suppose the number of "0" entries in G (with size w) is u right before a packet pkt with source s arrives ($s := pkt.src$ in line 10). Assume pkt belongs to a new flow and its flow label hashes to an index r. The value of $G[r]$ has value 0 with probability $\frac{u}{w}$. Therefore to obtain an unbiased estimator $\widehat{N_s}$ of the fan-out of the source s on the sampled traffic, we should statistically compensate for the fact that with probability $1 - \frac{u}{w}$, the bit $G[r]$ has value 1 and pkt will miss the update to L due to aforementioned hash collisions. It is intuitive that if we add $\frac{w}{u}$ to $\widehat{N_s}$, the resulting estimator is unbiased. To be more precise, suppose in a measurement epoch, the hash table is updated by altogether K packets $\{pkt_j,\ j = 1, 2, ..., K\}$ from a source s, whose flow labels hash to locations r_j's where $G[r_j] = 0$, and there are u_j 0's in G right before pkt_j arrives, respectively. The output of the hash table L, which is an unbiased estimator of the fan-out of s on the sampled traffic, is

$$\widehat{N_s} = \sum_{j=1}^{K} \frac{w}{u_j} \tag{1}$$

We show in the following lemma that this is an unbiased estimator of N_s and its proof can be found in [24].

Lemma 1 *$\widehat{N_s}$ is an unbiased estimator of N_s, i.e., $E[\widehat{N_s}] = N_s$.*

Then an unbiased estimator of the fan-out F_s of source s is given by scaling $\widehat{N_s}$ by $1/p$, i.e.,

$$\widehat{F_s} = \frac{1}{p} \sum_{j=1}^{K} \frac{w}{u_j} \tag{2}$$

where p is the sampling rate used in the flow sampling. We show in the following theorem that the estimator $\widehat{F_s}$ is unbiased. Its proof uses Lemma 1 and is also provided in [24].

Theorem 1 *$\widehat{F_s}$ is an unbiased estimator of F_s, i.e., $E[\widehat{F_s}] = F_s$.*

We now demonstrate that this solution will completely smooth out the aforementioned problem of traffic bursts , and allow the sampling rate to be close to the ratio between the hash table speed and the link rate, the theoretical upper limit in the worst case. The worst case for our scheme is that each flow contains only one packet (e.g., in the case of DDoS attacks)[6]. Even in this worst case, the update times to the hash table (viewed as a random process) is very close to Poisson[7] (nonhomogeneous as the value of u varies over time) since each new flow is sampled independently. Due to the "benign" nature of this arrival process, by employing a tiny SRAM buffer (e.g., holding 20 flow labels of 64 $\sim$ 100 bits each), a hash table that operates slightly faster than the average rate of this process will only miss a negligible fraction of updates due to buffer overflow. This process can be faithfully modeled as a Markov chain for rigorous analysis. We elaborate it with a numerical example in [24] and omit it here due to lack of space.

Notice that in Figure 2 the variable u, the number of "0" entries in G, decreases as more and more sampled flows are processed. When more and more packets pass through the data streaming module, u becomes small and hence the probability for a new flow to be recorded, $\frac{u}{w}$, decreases. Thereby the estimation error will increase. To maintain high accuracy, we specify a minimum value u_{min} for u. Once the value of u drops below u_{min}, the estimation procedure will use a new array (set to all "0"s initially) and start a new measurement epoch (with an empty hash table). Two sets of arrays and hash tables will be operated in an alternating manner so that the measurement can be performed without interruption. The parameter u_{min} is typically set to around $w/2$ (i.e., "half full").

3.3 Complexity analysis

The above scheme has extremely low storage (SRAM) complexity and allows for very high streaming speed.

Memory (SRAM) consumption. Each processed flow only consumes a little more than one bit in SRAM. Thus a reasonable amount of SRAM can support very high link speeds. For example, assuming the average flow size of 10 packets [11], 512KB SRAM is enough to support a measurement epoch which is slightly longer than 2 seconds for a link with 10 million packets per second even without performing any flow sampling. With 25% flow sampling which is typically set for OC-192 links the SRAM requirement is even brought down to 128KB.[8]

Streaming speed. Our algorithm in Figure 2 has two branches to deal with the packets arriving at the data streaming module. If the corresponding bit is "1", the packets only require one hash function computation and one read to SRAM. Otherwise they require one hash function computation, one read and one write (at the same location) to SRAM and an update to the hash table. Using efficient hardware implementation of hash function [18] and $5ns$

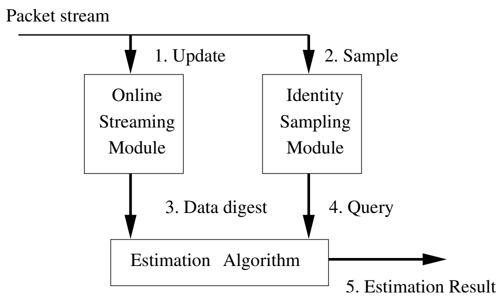


Figure 3: System model of the advanced scheme

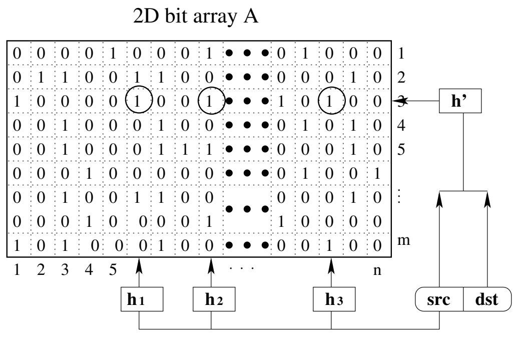


Figure 4: An instance of online streaming module

```
1. **Initialize**
2.    $A[i,j] := 0, \quad i = 1, 2, ..., m \quad j = 1, 2, \cdots, n$

3. **Update**
4.    Upon the arrival of a packet $pkt$
5.      $row := h'(< pkt.src, pkt.dst >)$
6.      for $i := 1$ to $k$
7.        $col := h_i(pkt.src)$
8.        $A[row, col] := 1$
```

Figure 5: Algorithm of online streaming module

SRAM, all operations in the data streaming module can be finished in 10's of ns in both cases.

3.4 Accuracy analysis

Now we establish the following theorem to characterize the variance of the estimator $\widehat{F_s}$ in Formula 2. Its proof can be found in [24]

Theorem 2

$$Var[\widehat{F_s}] \approx \frac{\sum_{j=1}^{pF_s} \frac{w-u_j}{u_j}}{p^2} + \frac{F_s(1-p)}{p}$$

Remark: The above variance consists of two terms. The first term corresponds to the variance of the error term in estimating the sampled fan-out, scaled by $\frac{1}{p^2}$ (to compensate for sampling), and the second term corresponds to the variance of the error term in inverting flow sampling process. Since these two errors are approximately orthogonal to each other, their total variance is the sum of their individual variances.

4 The advanced scheme

In this section we propose the advanced scheme that is more sophisticated than the simple scheme but can offer more accurate fan-out estimations. It is based on the aforementioned design methodology of separating identity gathering from counting. Its system model is shown in Figure 3. There are two parallel modules processing the incoming packet stream. The data streaming module encodes the fan-out information for each and every source (arc 1 in Figure 3) into a very compact data structure, and the identity sampling module captures the candidate source identities which have potential to be super sources (arc 2). These source identities are then used by an *estimation algorithm* to look up the data structure (arc 3) produced by the data streaming module (arc 4) to get their corresponding fan-out estimates. The design of these modules are described in the following subsections.

4.1 Online streaming module

The data structure used in the online streaming module is quite simple: an $m \times n$ 2-dimensional bit array A. The bits in A are set to all "0"s at the beginning of each measurement epoch. The algorithm of updating A is shown in Figure 5. Upon the arrival of a packet pkt, $pkt.src$ is hashed by k independent[9] hash functions $h_1, h_2, \cdots, h_k$ with range $[1..n]$. The hashing results $h_1(pkt.src)$, $h_2(pkt.src)$, ..., $h_k(pkt.src)$ are viewed as column indices into A. In our scheme, k is set to 3, and the rationale behind it will be discussed in Section 4.3. Then, the flow label $< pkt.src, pkt.dst >$ is hashed by another independent hash function h' (with range $[1..m]$) to generate a row index of A. Finally, the k bits located at the intersections of the selected row and columns are set to "1". An example is shown in Figure 4, in which the three intersection bits (circled) are set to "1"s. When A is filled (by "1") to a threshold percentage we terminate the current measurement epoch and start the decoding process[10]. In Section 4.2, we show that the above process produces a very compact and accurate (statistical) encoding of the fan-outs of the sources, and present the corresponding decoding algorithm.

Readers may feel that the above 2D bit array A is a variant of Bloom filters [1]. This is not the case since although its encoding algorithm is similar to that of a Bloom filter (flipping some bits to "1"s), we decode the 2D bit array for a different kind of information (the fan-out count) than if we really use it as a Bloom filter (check if a particular source-destination pair has appeared). Our encoding algorithm is not well engineered for being used as a Bloom filter. And our decoding algorithm, shown next, is also fundamentally different from that of a Bloom filter.

The proposed online streaming module has very low memory consumption and high streaming speed:

Memory (SRAM) consumption. Our scheme is extremely memory-efficient. Each source-destination pair (flow) will set 3 bits in the bit vectors to "1"s and consume a little more than 3 bits of SRAM storage[11]. We will show that the scheme provides very high accuracy using reasonable amount of SRAM (e.g., 128KB) in Section 6.

Streaming speed. Each update requires only 4 hash function computations and 3 writes to the SRAM. We require that these four hash functions are independent and amendable to hardware implementation. They can be chosen from the H_3 hash function family [2, 18], which, with hardware implementation, can produce a hash output within a few nanoseconds. Then with commodity $5ns$ SRAM our scheme would allow around 40 million packets per second, thereby supporting 40 Gbps traffic stream assuming a conservative average packet size of 1,000 bits.

4.2 Estimation module

For each source identity recorded by the sampling module (described later), the estimation module decodes its approximate fan-out from the 2D bit array A, the output of the data streaming module. In this section, we describe this decoding algorithm in detail.

When we would like to know F_s, the fan-out of the source s, s is hashed by the hash functions $h_1, \cdots, h_k$, which are defined and used in the online streaming module, to obtain k column indices. Let A_i, $i = 1, 2, \cdots, k$, be the corresponding columns (viewed as bit vectors). In the following, we derive, step by step, an accurate and almost unbiased estimator of F_s, as a function of A_i, $i = 1, 2, \cdots, k$.

Let the set of packets hashed into column A_i during the corresponding measurement epoch be T_i and the number of bits in A_i that are "0"s be U_{T_i}. Note that the value U_{T_i} is a part of our observation since we can obtain U_{T_i} from A_i through simple counting, although the notation itself contains T_i, the size of which we would like to estimate. Recall the size of the column vector is m. A fairly accurate estimator of $|T_i|$, the number of packets of T_i, adapted from [21], is

$$D_{T_i} = m \ln \frac{m}{U_{T_i}} \tag{3}$$

Note that F_s, the fan-out of the source s, is equal to $|T_1 \cap T_2 \cap \cdots \cap T_k|$, if during the measurement epoch, no other sources are hashed to the same k columns $A_1, A_2, \cdots, A_k$. Otherwise $|T_1 \cap T_2 \cap \cdots \cap T_k|$ is the sum of the fan-outs of all (more than 1) the sources that are hashed into $A_1, A_2, \cdots, A_k$. We show in the next section, that the probability with which the latter case happens is very small when $k = 3$. We obtain the following estimator of F_s, which is in fact derived as an estimator for $|T_1 \cap T_2 \cap \cdots \cap T_k|$.

$$\begin{aligned}\widehat{F_s} = &\sum_{1 \le i \le k} D_{T_i} - \sum_{1 \le i_1 < i_2 \le k} D_{T_{i_1} \cup T_{i_2}} \\ &+ \sum_{1 \le i_1 < i_2 < i_3 \le k} D_{T_{i_1} \cup T_{i_2} \cup T_{i_3}} \\ &+ \cdots + (-1)^{k-1} D_{T_1 \cup T_2 \cdots \cup T_k}\end{aligned} \tag{4}$$

Here $D_{T_i \cup \cdots \cup T_j}$, is defined as $m \ln \frac{m}{U_{T_i \cup \cdots \cup T_j}}$, where $U_{T_i \cup \cdots \cup T_j}$ denotes the number of "0"s in the bit vector $B_{T_i \cup \cdots \cup T_j}$ which is the result of hashing the set of packets $T_i \cup \cdots \cup T_j$ into a single empty bit vector. The bit vector $B_{T_i \cup \cdots \cup T_j}$ is computed as the bitwise-OR of $A_i, \ldots, A_j$. One can easily verify the correctness of this computation with respect to the semantics of $B_{T_i \cup \cdots \cup T_j}$.

We need to show that the RHS of Formula 4 is a fairly good estimator of $|T_1 \cap T_2 \cap \cdots \cap T_k|$. Note that

$$\begin{aligned}|T_1 \cap T_2 \cap \cdots \cap T_k| = &\sum_{1 \le i \le k} T_i - \sum_{1 \le i_1 < i_2 \le k} T_{i_1} \cup T_{i_2} \\ &+ \sum_{1 \le i_1 < i_2 < i_3 \le k} T_{i_1} \cup T_{i_2} \cup T_{i_3} \\ &+ \cdots + (-1)^{k-1} T_1 \cup T_2 \cdots \cup T_k\end{aligned} \tag{5}$$

by the principle of inclusion and exclusion. Since $D_{T_i \cup \cdots \cup T_j}$ is a fairly good estimator of $|T_i \cup \cdots \cup T_j|$ according to [21], we obtain the RHS of Formula 4 by replacing the terms $|T_i \cup \cdots \cup T_j|$ in Formula 5 by $D_{T_i \cup \cdots \cup T_j}$, $1 \le i < j \le k$. Note that it is not correct to directly use the bitwise-AND of $A_1, A_2, ..., A_k$ for estimating $|T_1 \cap T_2 \cap \cdots \cap T_k|$ using Formula 3, because the bit vector corresponding to the result of hashing the set of packets $T_1 \cap T_2 \cap ... \cap T_k$ into an empty bit vector, is not equivalent to the bitwise-AND of $A_1, ..., A_k$.

The estimator in Formula 4 generalizes the result in [21] which is developed for the special case $k = 2$. We will show that our scheme only needs to use the special case of $k = 3$, which is

$$\begin{aligned}\widehat{F_s} = &D_{T_1} + D_{T_2} + D_{T_3} - D_{T_1 \cup T_2} - D_{T_1 \cup T_3} - D_{T_2 \cup T_3} \\ &+ D_{T_1 \cup T_2 \cup T_3}\end{aligned} \tag{6}$$

The computational complexity of estimating the fan-out of a source is dominated by $2^k - k - 1$ bitwise-OR operations among k column vectors. Such vectors can be encoded as one or more unsigned integers so that the bit-parallelism can significantly reduce the execution time. Since m is typically 64 bits in our scheme, the whole vector can be held in two 32-bit integers. Therefore, in our scheme where $k = 3$, estimation of the fan-out of each source only needs 8 bitwise-OR operations between 32-bit integers. We also need to count the number of "0"s in a vector (to get U_T values). This can be sped up significantly by using a pre-computed table (in SRAM) of size 262,144 ($= 2^{16} \times 4$) bits that stores the number of "0"s in all 16-bit numbers. Our estimation of the execution time shows that our scheme is fast enough to support OC-768 operations.

4.3 Accuracy analysis

In this section we first briefly explain the rationale behind setting k to 3 in the estimator and then analyze the accuracy of our estimator rigorously. We set k (the number of "column" hash functions) to 3 due to the following two considerations. First, we mentioned before that if two sources s_1 and s_2 both are hashed to the same k columns, our decoding algorithm will give us an estimate of their total fan-out, when we use s_1 or s_2 to lookup the 2D array. We certainly would like the probability with which this scenario occurs to be as small as possible. This can be achieved by making k as large as possible. However, larger k implies larger computational and storage complexities at the online streaming module. We will show that $k = 3$ makes the probability of the aforementioned hash collision very small, and at the same time keeps the computational and storage complexities of our scheme modest.

Now we derive η, the probability that at least two sources happen to hash to the same set of k columns by h_1, h_2, ..., h_k. It is not hard to show, using straightforward combinatorial arguments, that $\eta = 1 - \frac{\binom{n^k}{S} S!}{n^{kS}}$, where S is the total number of the distinct sources during the measurement epoch. We observe that, given typical values for n and S, η is quite large when $k = 2$, but drops to a very low value when $k = 3$. For example, when $n = 16$K and $S = 100,000$, η is close to 1 when $k = 2$, but drops to 0.002 when $k = 3$.

The following theorem characterizes the variance of the estimator in Formula 6, which is also its approximate mean square error (MSE), since the estimator is almost unbiased and the impact of η (discussed above) on the estimation error is very small when $k = 3$. Its proof can be found in [24]. This is an extension of our previous variance analysis in [23] which is derived for the special case $k = 2$. Let t_T denote $|T|/m$, which is the load factor of the bit vector(of size m) when the corresponding set T of source-destination pairs are hashed into it, in the following theorem.

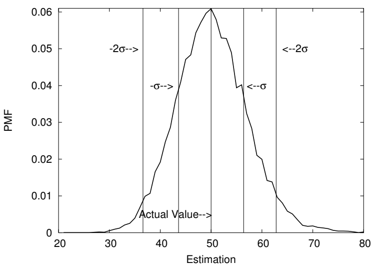


Figure 6: Distribution of the estimation from Monte-Carlo simulation ($m = 64$b, $k = 3$ and $F_s = 50$). σ is the standard deviation computed from Theorem 3

Theorem 3 *The variance of $\widehat{F_s}$ is given by*

$$
\begin{aligned}
Var[\widehat{F_s}] \approx & -m\sum_{i=1}^{3} f(t_{T_i}) - m \sum_{1\le i_1<i_2\le 3} f(t_{T_{i_1}\cup T_{i_2}}) \\
& + 2m(f(t_{T_1\cup(T_2\cap T_3)}) + f(t_{T_2\cup(T_1\cap T_3)}) + f(t_{T_3\cup(T_2\cap T_1)})) \\
& + 2m \sum_{1\le i_1<i_2\le 3} f(t_{T_{i_1}\cap T_{i_2}}) \\
& - 2m(f(t_{T_1\cap(T_2\cup T_3)}) + f(t_{T_2\cap(T_1\cup T_3)}) + f(t_{T_3\cap(T_2\cup T_1)})) \\
& + mf(t_{T_1\cup T_2\cup T_3})
\end{aligned}
$$

where $f(t) = e^t - t - 1$.

An example distribution of $\widehat{F_s}$ with respect to the actual value F_s is shown in Figure 6. We obtained this distribution with 20,000 runs of the Monte-Carlo simulations with the following parameters. In this empirical distribution, the actual fan-out F_s is 50, the size of the column vector m is 64 bits. The load factor is set to $t_{T_i} = 1.5$ for $1 \le i \le 3$. Also, since the sets T_1, T_2, and T_3 have 50 flows in common, we set $t_{T_i \cap T_j} = \frac{50}{64} = 0.78125$, for $1 \le i < j \le 3$. Here we implicitly assume that pairs of them do not have any flows in common other than these 50 flows, which happens with very high probability ($= 1 - \frac{1}{n^2} + \frac{1}{n^3}$) anyway given a large n (e.g., 16K). In this example the standard deviation σ is around 6.4 as computed from Theorem 3. We observe that the size of the tail that falls outside 2 standard deviations from the mean is very small ($< 4\%$). This shows that, with high probability, our estimator will not deviate too much from the actual value. In Section 6, the trace-driven experiments on the real-world Internet traffic will further validate this.[12]

Note that given the size of m, our scheme is only able to accurately estimate fan-out values up to $m \ln m + O(m)$, because if the actual fan-out F_s is much larger than that, we will see all 1's in the corresponding column vectors with high probability (due to the result of the "coupon collector's problem" [13]). In this case, the only information we can obtain about F_s is that it is no smaller than $m \ln m$. Fortunately, for the purpose of detecting super sources, this information is good enough for us to declare s a super source, as long as the threshold for super sources is much smaller than $m \ln m$. However, in some applications (e.g., estimating the spreading speed of a worm), we may also want to know the approximate fan-out value. This can be achieved using a *multi-resolution* extension of our advanced scheme. The methodology of *multi-resolution* is quite standard and has been used in several recent works [6, 12, 11]. The extension in our context is straightforward. We omit its detailed specifications here in interest of space.

4.4 Identity sampling module

The purpose of this module is to capture the identities of potential super sources that will be used to look up the 2D array to get their fan-out estimations. The filtering after sampling technique proposed in Section 3 is adopted here with a slightly different recording strategy. Instead of constructing a hash table to record the sources and their fan-out estimation, here we only record the source identities sequentially in the DRAM. Since this strategy avoids expensive hash table operations and sequential writes to DRAM can be made very fast (using burst mode), very high sampling rate can be achieved. With commodity $5ns$ SRAM and $60ns$ DRAM, this recording strategy will be able to process more than 12.5 million packets per second. At this speed, we can record 100% and 25% flow labels for OC-192 and OC-768 links respectively. With such a high sampling rate, the probability that the identity of a real super source misses sampling is very low. For example, given 25% sampling rate the probability that a source with fan-out 50 fails to be recorded is only 5.6×10^{-7} (=$(1 - 25\%)^{50}$).

5 Estimating outstanding fan-outs

In this section we describe how to extend the advanced scheme to detect the sources that have contacted but have not obtained acknowledgments from a large number of distinct destinations (i.e., the sources with large outstanding fan-outs). Although both of our schemes have the potential to support this extension we focus on the advanced scheme in this work and leave the extension of the simple scheme for future research. In the following sections we show how to slightly modify the operations of the online streaming module and the estimation module of the advanced scheme for estimating outstanding fan-outs. The sampling module does not need to be modified.

5.1 Online streaming module

The online streaming module employs two 2D bit arrays A and B of identical size and shape. The array A encodes the fan-outs of sources in traffic in one direction (called "outbound") in the same way as in the advanced scheme (shown in Figure 5). The array B encodes the fan-ins of the destinations of the acknowledgment packets in the opposite direction (called "inbound"). Its encoding algo-

```
1. **Initialize**
2.    $B[i, j] := 0, i = 1, 2, ..., m \quad j = 1, 2, \cdots, n$

3. **Update**
4.    Upon the arrival of a packet $pkt$
5.      **if** $pkt$ is an acknowledgment packet
6.        $row := h'(< pkt.dst, pkt.src >)$
7.        **for** $i$ := 1 to $k$
8.          $col := h_i(pkt.dst)$
9.          $B[row, col] := 1$
```

Figure 7: Algorithm for updating the 2D bit array B to record ACK packets

rithm is shown in Figure 7. It is a transposed version of the algorithm shown in Figure 5 in the sense that all occurrences of "$pkt.src$" are replaced with with "$pkt.dst$" and "$pkt.dst$" with "$pkt.src$". This transposition is needed since a source in the outbound traffic appears in the inbound acknowledgment traffic as a destination, and after transposing two packets that belong to a flow and its "acknowledgment flow" respectively will result in a write of "1" to the same bit locations in A and B respectively. This allows us to essentially take a "difference" between A and B to obtain the decoding of outstanding fan-outs of various sources, shown next.

5.2 Estimation module

We compute the bitwise-OR of A and B, denoted as $A \vee B$. For each source s, we decode its fan-out from $A \vee B$ using the same decoding algorithm as described in Section 4.2. Similarly, we decode its fan-in in the acknowledgment traffic from B. Our estimator of the outstanding fan-out of s is simply the former subtracted by the latter.

Now we explain why this estimator will provide an accurate estimate of the outstanding fan-out of a source s. Let S_1 be the set of flows whose source is "s" in the outbound traffic. Let S_2 be the set of flows whose destination is "s" in the inbound acknowledgment traffic. Clearly the quantity we would like to estimate is simply $|S_1 - S_2|$. The correctness of our estimator is evident from the following three facts: (a) $|S_1 - S_2|$ is equal to $|S_1 \bigcup S_2| - |S_2|$; (b) decoding from $A \vee B$ will result in a fairly accurate estimate of $|S_1 \bigcup S_2|$ and (c) decoding from B will result in a fairly accurate estimate of $|S_2|$.

6 Evaluation

In this section, we evaluate the proposed schemes using real-world Internet traffic traces. Our experiments are grouped into three parts corresponding to the three algorithms presented: the simple scheme, the advanced scheme, and its extension to estimate outstanding fan-outs. The experimental results show that our schemes allow for accu-

Trace	# of sources	# of flows	# of packets
IPKS$+$	119,444	151,260	1,459,394
IPKS$-$	96,330	125,126	1,655,992
USC	84,880	106,626	1,500,000
UNC	55,111	101,398	1,495,701

Table 1: Traces used in our evaluation. Note that source is $< src\_IP, src\_port >$, destination is dst_IP and flow label is the distinct source-destination pair.

rate estimation of fan-outs and hence the precise detection of super sources.

6.1 Traffic Traces and Flow definitions

Trying to make the experimental data as representative as possible, we use packet header traces gathered at three different locations of the Internet, namely, University of North Carolina (UNC), University of Southern California (USC), and NLANR. The trace form UNC was collected on a 1 Gbps access link connecting the campus to the rest of the Internet, on Thursday, April 24, 2003 at 11:00 am. The trace from USC was collected at their Los Nettos tracing facility on Feb. 2, 2004. We also use a pair of unidirectional traces from NLANR: $IPKS+$ and $IPKS-$, collected simultaneously on both directions of an OC192c link on June 1, 2004. The link connects Indianapolis (IPLS) to Kansas City (KSCY) using Packet-over-SONET. This pair of traces is especially valuable to evaluate the extended advanced scheme for estimating outstanding fan-outs. All the above traces are either publicly available or available for research purposes upon request. Table 1 summarizes all the traces used in the evaluation. We will use USC, UNC and $IPKS+$ to evaluate our simple scheme and advanced scheme and use the concurrent traces $IPKS+$ and $IPKS-$ to evaluate the extension.

As mentioned before, a source/destination label can be any combination of source/destination fields from the IP header. Two different definitions of source and destination labels are used in our experiments, targeting different applications. In the first definition, source label is the tuple <src_IP, src_port> and destination label is <dst_IP>. This definition targets applications such as detecting worm propagation and locating popular web servers. The flow statistics displayed in Table 1 use this definition. In the second definition, Second, source label is <src_IP> and destination label is the tuple <dst_IP, dst_port>. This definition targets applications such as detecting infected sources that conduct port scans. The experimental results presented in this section use the first definition of source and destination labels unless noted otherwise.

6.2 Accuracy of the simple scheme

In this section, we evaluate the accuracy of the simple scheme in estimating the fan-outs of sources and in detecting super sources. Figure 8 compares the fan-outs of the sources estimated using our simple scheme with the their actual fan-outs in traces $IPKS+$, UNC, and USC respectively. In these experiments, a flow sampling rate of $1/4$ and a bit array of size $128K$ bits is used. The figure only plots the points whose actual fan-out values are above 15 since lower values (i.e., < 15) are not interesting for finding super sources. The solid diagonal line in each figure denotes perfect estimation, while the dashed lines denote an estimation error of $\pm 15\%$. The dashed lines are parallel to the diagonal line since both x-axis and y-axis are on the log scale. Clearly the closer the points cluster around the diagonal, the more accurate the estimation is. We observe that the simple scheme achieves reasonable accuracy for relatively large fan-outs in all three traces. Figure 8 also reflects the false positives and negatives in detecting super sources. For a given threshold 50, the points that fall in "Area I" corresponds to *false positives*, i.e., the sources whose actual fan-outs are less than the threshold but the estimated fan-outs are larger than the threshold. Similarly, the points that fall in "Area II" corresponds to *false negatives*, i.e., the sources whose actual fan-outs are larger than the threshold but the estimated fan-outs are smaller than the threshold. We observe that in Figure 8, points rarely fall into Areas I and II (i.e., very few false positives and negatives[13]).

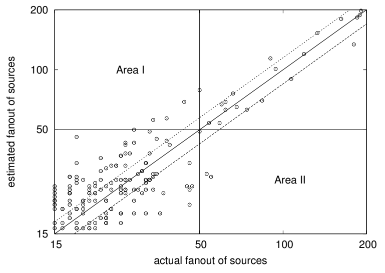
While this scheme works well with 1/4 sampling rate, it cannot produce good estimations with smaller sampling rates (e.g., 1/16. We omit the experimental results here due to lack of space.). However such lower sampling rates might be necessary to keep up with very high link speeds such as 40 Gbps (OC-768).

We repeat the above experiment under the aforementioned second definition of source and destination, in which the source label is <src_IP> and destination label is <dst_IP, dst_port>. Figure 9 plots the estimated fan-outs of sources in trace IPKS+. With this definition the trace $IPKS+$ has 9,359 sources and 140,140 distinct source-destination pairs. We can see from the figure that our estimation is also quite accurate with this second definition of source and destination.

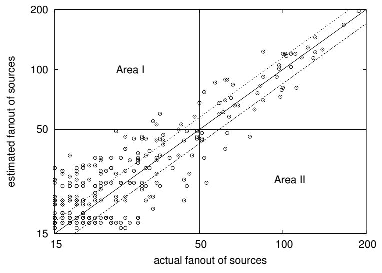
6.3 Accuracy of the advanced scheme

In this section we evaluate the accuracy of the advanced scheme using both trace-driven simulation and theoretical analysis. The estimation accuracy of the advanced scheme is a function of the various design parameters, including the size and shape of the 2D bit array A (i.e., the number of rows m and columns n) and the number of hash functions (k).

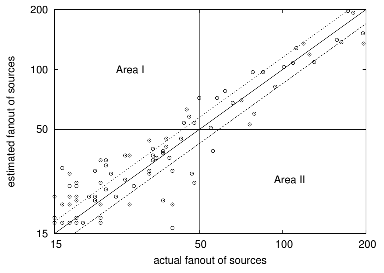
In the experiments we set the size of A to 128KB (64 rows $\times$ 16,384 columns), $k = 3$ and the flow sampling rate to 1. This configuration is very space-efficient. For example it only uses 7 bits per flow on the average for the



(a) trace IPKS+



(b) trace UNC



(c) trace USC

Figure 8: Actual vs. estimated fan-outs of sources by the simple scheme given the flow sampling rate 1/4. Notice both axes are on logscale.

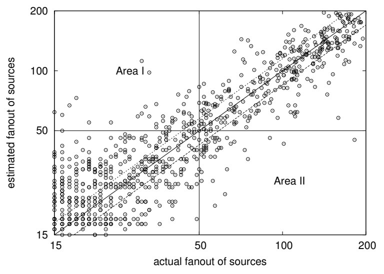


Figure 9: Actual vs. estimated fan-out of sources for trace $IPKS+$ with flow sampling rate 1/4. The aforementioned second definition of source and destination labels is used here. Note that both x-axis and y-axis are on logscale.

trace $IPKS+$.

6.3.1 Trace-driven experiments

Figure 10 compares the fan-out values estimated using the advanced scheme with the actual fan-outs of the corresponding sources given three different traces. Compared with the corresponding plots in Figure 8, the points are much closer to the diagonal lines, which means that the advanced scheme is much more accurate than the simple scheme.

In Figure 11, we repeat the experiments with source and destination labels defined as <src_IP> and <dst_IP,dst_port>, respectively. Compared with the result of the simple scheme (Figure 9) the points are much closer to the diagonal again, indicating the higher accuracy achieved by the advanced scheme.

Note that in the experiments above we set the flow sampling rate to 1 instead of $1/4$ used in the experiments of the simple scheme since as we described in Section 3.3 and Section 4.4 respectively for a fully utilized OC-192 link the simple scheme requires $1/4$ flow sampling rate but the identity sampling module of the advanced scheme can record 100% flow labels.

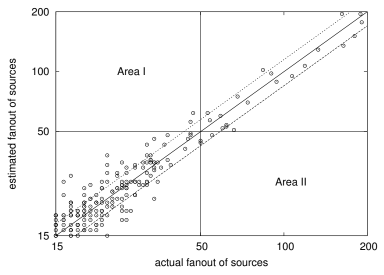
6.3.2 Theoretical accuracy

The accuracy of the estimation can be characterized by the average relative error of the estimator, which is equal to the standard deviation of the ratio $\frac{\widehat{F_s}}{F_s}$ which can be computed by Theorem 3.

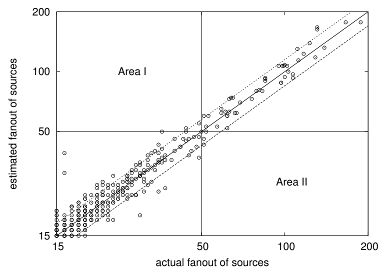
Figure 12 shows the average relative error plotted against estimated fan-outs for the sources in the trace $IPKS+$. Experiments on other traces produced similar results. The average relative error shows a sharply downward trend when the estimated value of fan-out increases in Figure 12. This is a very desirable property as we would like our mechanism to be more accurate when estimating larger fan-outs. Towards the right extreme of the figure, the average relative error starts to increase. This is because the selected bit vectors become almost full ("saturation") when the fan-out value is close to 266 ($m \ln m$). As we discussed in Section 4.3 the accuracy of our estimator would degrade when the corresponding column vectors become saturated[14]. It does not affect the accuracy of our scheme for detecting super sources, but to accurately estimate the exact fan-out values that are large, the aforementioned multi-resolution extension [6, 11, 12] is needed.

The accuracy of the estimator can also be characterized by the probability of the estimated values $\widehat{F_s}$ falling into the interval $[(1-\epsilon)F_s, (1+\epsilon)F_s]$, where F_s is actual fan-out of the source s. This quantity can be numerically computed by Monte-Carlo Simulation as follows. We first use the trace UNC to construct the 2D bit array A (serving as "background noise"). Then we synthetically generate a source that has fan-out value F_s and insert it into A by randomly selecting 3 different columns. The estimator (Formula 6) is used to obtain the $\widehat{F_s}$. The above operations are repeated 100,000 times to compute the probabilities shown in Figure 12.

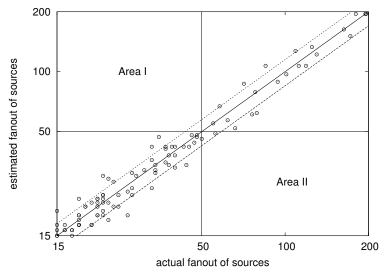
Figure 13 shows the plot of $(1-\delta)$ for different values of F_s, where $1-\delta = Prob[(1-\epsilon)F_s \leq \widehat{F_s} \leq (1+\epsilon)F_s]$. Each curve corresponds to a specific level of relative



(a) trace IPKS+



(b) trace UNC



(c) trace USC

Figure 10: Actual vs. estimated fan-out of sources by the advanced scheme. Notice both axes are on logscale.

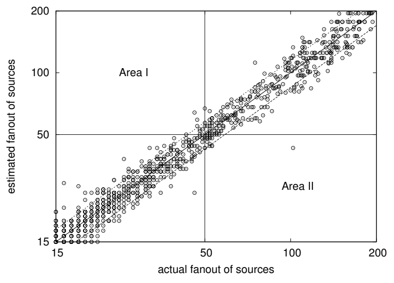


Figure 11: Actual vs. estimated fan-out of sources for trace $IPKS+$ under the second flow definition by the advanced scheme. Notice both axes are on logscale.

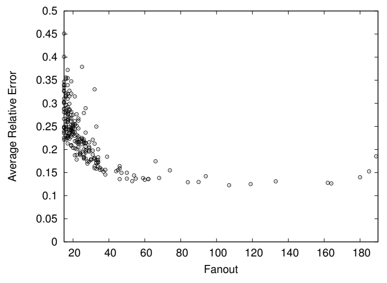


Figure 12: Average relative error for various fan-out values in the trace $IPKS+$.

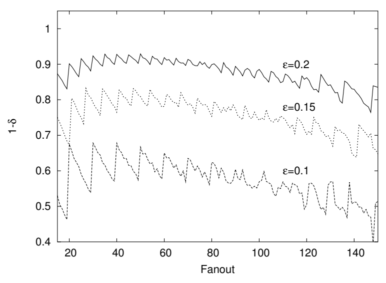


Figure 13: Probability that the estimate $\widehat{F_S}$ is within a factor of $(1\pm\epsilon)$ of the actual fan-out F_s for various values of ϵ.

error tolerance, i.e., a specific choice of ϵ, and represents the probability that the estimated value is within this factor of the actual value. For example, the curve for $\epsilon = 0.2$ shows that around 85% of the time the estimate is within 20% of the actual value. Notice how the curves in the figure have an upward trend first and then show a downward trend as the fan-out increases further. This corresponds exactly to the aforementioned "saturation" situation.

6.4 Accuracy of the extension to estimate outstanding fan-outs

To evaluate the extension of the advanced scheme to estimate outstanding fan-outs we use the pair of traces, $IPKS+$ and $IPKS-$, collected simultaneously on both directions of a link. We extract all the acknowledgment packets from $IPKS-$ to produce the 2D bit array B using the transposed update algorithm (Figure 7). The same parameters are configured for both 2D bit arrays A and B. Figure 14 shows the scatter diagram of the fan-out estimated using our proposed scheme (y axis) vs. actual outstanding fan-out (x axis). The fact that most points are concentrated within a narrow band of fixed width along the diagonal line indicates that our estimator is accurate on estimating outstanding fan-outs.

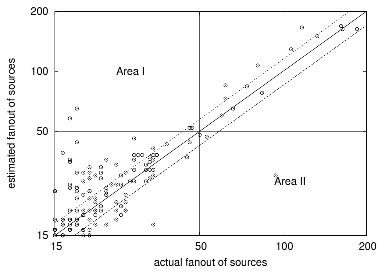


Figure 14: Actual vs. estimated fan-out of sources by extension of the advanced scheme including deletions. Notice both axes are on logscale.

7 Related work

The problem of detecting super sources and destinations has been studied in recent years. In general, three approaches have been proposed in the literature:

1. A straightforward approach is to keep track, for each source/destination, the set of distinct destinations/sources that it contacts, using a hash table. This approach is adopted in Snort [19] and FlowScan [17]. It is straightforward to implement but not memory-efficient, since most of the source-destination pairs in the hash table do not come from super sources/destinations. As mentioned before, this approach is not feasible for monitoring high-speed links since the hash table typically can only fit into DRAM.

2. Data streaming algorithms are designed by Estan et al. [6] mainly for estimating the number of active flows in the Internet traffic. However, it is stated in [6], that one variant of their scheme, i.e., triggered bitmap, can be used for identifying the super sources. This algorithm maintains a small bitmap (4 bytes) for each source (subject to hash collision), for estimating its fan-out. Once the number of bits set in the small bitmap exceeds a certain threshold (indicting a large fan-out), a large multi-resolution bitmap is allocated to perform a more accurate counting of its fan-out. Since the implementation of the binding between the source and the bitmap is not elaborated in [6], we speculate that the binding is implemented as a hash table, which can be quite costly if it has to fit in SRAM (for high-speed processing). Also, its memory efficiency is further limited by allocating at least 4 bytes for each source.

3. Recently Venkataraman et al. [20] propose two flow sampling based techniques for detecting super sources/destinations. Their one-level and two-level filtering schemes both use a traditional hash-based flow sampling technique for estimating fan-outs. We explained in Section 3.1 that, when this scheme is used for high-speed links (e.g., 10 or 40 Gbps), the sampling rate is typically low due to the aforementioned traffic burst problem. This prevents the algorithms from achieving high estimation accuracy. In addition, the memory usage of both schemes, which use hash tables, is much higher than our advanced scheme. They only mentioned the possibility of replacing hash table with Bloom filters to save space, but did not fully specify the details of the scheme (e.g., parameter settings). This makes a head-on comparison of our schemes with theirs very difficult. In fact, after this replacement (of hash table with Bloom filters), their scheme becomes a variant of Space Code Bloom Filter (SCBF) we proposed in [12], with a slightly different decoding algorithm[15]. Their decoding algorithm has similar computational complexity as that of SCBF, which is an order magnitude more expensive than that of our advanced scheme. This may prevent our SCBF scheme (and their scheme as well) from operating at very high link speeds (e.g., 40 Gbps).

8 Conclusion

Efficient and accurate detection of super sources and destinations at high link speeds is an important problem in many network security and measurement applications. In this work we attack the problem with a new insight that sampling and streaming are often suitable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is an excellent way to recover the complete information. This insight leads to two novel methodologies of combining the power of streaming and sampling, namely, "filtering after sampling" and "separation of counting and identity gathering", upon which our two solutions are built respectively. The first solution improves the estimation accuracy of hash-based flow sampling by allowing for much higher sampling rate, through the use of a embedded data streaming module for filtering/smoothing the bursty incoming traffic. Our second solution combines the power of data streaming in efficiently retaining and estimating fan-out/fan-in associated with a given source/destination, and the power of sampling in generating a list of candidate source/destination identities. Mathematical analysis and trace-driven experiments on real-world Internet traffic show that both solutions allow for accurate detection of super sources and destinations.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7):422–426, 1970.

[2] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, pages 143–154, 1979.

[3] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE transaction of Networking*, pages 280–292, June 2001.

[4] N. Duffield, C. Lund, and M. Thorup. Estimating flow distribution from sampled flow statistics. In *Proc. ACM SIGCOMM*, August 2003.

[5] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. ACM SIGCOMM*, August 2002.

[6] C. Estan and G. Varghese. Bitmap algorithms for counting active flows on high speed links. In *Proc. ACM/SIGCOMM IMC*, October 2003.

[7] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proc. IEEE GLOBECOM*, December 1999.

[8] N. Hohn and D. Veitch. Inverting sampled traffic. In *Proc. ACM/SIGCOMM IMC*, October 2003.

[9] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdn and web sites. In *Proc. World Wide Web Conference*, May 2002.

[10] R. Karp, S. Shenker, and C. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28:51–55, 2003.

[11] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proc. ACM SIGMETRICS*, 2004.

[12] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-Code Bloom Filter for Efficient per-flow Traffic Measurement. In *Proc. IEEE INFOCOM*, March

2004.

[13] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[14] CISCO Tech Notes. Cisco netflow. available at http://www.cisco.com/warp/public/732/netflow/index.html.

[15] V. Paxon. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review*, 2001.

[16] D. S. Phatak and T. Goff. A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments. In *Proc. IEEE INFOCOM*, June 2002.

[17] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *Proc. USENIX LISA*, 2000.

[18] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, pages 1378–1381, 1997.

[19] M. Roesch. Snort–lightweight intrusion detection for networks. In *Proc. USENIX Systems Administration Conference*, 1999.

[20] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Proc. NDSS*, 2005.

[21] K.Y. Whang, B.T. Vander-zanden, and H.M. Taylor. A linear-time probabilistic counting algorithm for database applications. *IEEE transaction of Database Systems*, pages 208–229, June 1990.

[22] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application. In *Proc. ACM/SIGCOMM IMC*, October 2004.

[23] Q. Zhao, A. Kumar, J. Wang, and J. Xu. Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. In *Proc. ACM SIGMETRICS*, June 2005.

[24] Q. Zhao, A. Kumar, and J. Xu. Joint data streaming and sampling techniques for detection of super sources and destinations. In *Technical Report*, July 2005.

Notes

1. Super sources have also been referred to as "*superspreaders*" in literature [20].

2. As a note of clarification, the term *data streaming* here has no connection with the transmission of multimedia data known as media (audio and video) streaming [16].

3. There is no explicit inversion procedure to recover the number of flows if packet sampling is used. The technique used in [4] may be helpful but does not provide accurate answers.

4. A small buffer in SRAM will not be able to smooth out such bursts since at high link speeds, such bursts can easily fill up several Megabytes of buffer in a matter of milliseconds.

5. We can use multiple independent hash functions to reduce the probability of collisions. But it will significantly increases the overhead of updating G and does not improve the estimation result too much.

6. Note that the worst case for hash-based flow sampling is different. It occurs when a few of the sampled flows contain most of the traffic on a link.

7. The inter-arrival time is in fact of geometric distribution.

8. We assume a conservative average packet size of 1,000 bits, to our disadvantage. Measurements from real-world Internet traffic report much larger packet sizes.

9. Such hash functions are referred to as k-universal hash function in literature [2]. It has been shown empirically in [2] that the H_3 family of hash functions are very close to k-universal statistically when operating on real-world data, for small k values (e.g., $k \leq 4$).

10. Again, two ping-pong modules can be used in an alternating fashion to avoid any operational interruption.

11. This is estimated based on the typical *load factor* (defined later) we place on the bit vector.

12. Note that we do not show an example distribution for the previous simple scheme since the estimator $\widehat{F_s}$ of it relies on where the flows with source s appear in the packet stream, i.e., the values of u_j when the flows arrive (cf. Formula 2). Therefore the estimator may have different distributions given a fixed value of F_s.

13. One shall not simply compare this false positive and negative ratios with the results in [20] since there only when the scheme fails to detect a source whose fan-out is several (say 5) times larger than the threshold will a false negative be declared.

14. For more details about this please refer to [21].

15. In [12], we decode for the exact value of the parameter to be estimated while their scheme [20] decodes for a lower bound of the parameter.

Optimal Combination of Sampled Network Measurements

Nick Duffield Carsten Lund Mikkel Thorup
AT&T Labs–Research, 180 Park Avenue, Florham Park, New Jersey, 07932, USA
{duffield,lund,mthorup}research.att.com

Abstract

IP network traffic is commonly measured at multiple points in order that all traffic passes at least one observation point. The resulting measurements are subsequently joined for network analysis.

Many network management applications use measured traffic rates (differentiated into classes according to some key) as their input data. But two factors complicate the analysis. Traffic can be represented multiple times in the data, and the increasing use of sampling during measurement means some classes of traffic may be poorly represented.

In this paper, we show how to combine sampled traffic measurements in way that addresses both of the above issues. We construct traffic rate estimators that combine data from different measurement datasets with minimal or close to minimal variance. This is achieved by robust adaptation to the estimated variance of each constituent. We motivate the method with two applications: estimating the interface-level traffic matrix in a router, and estimating network-level flow rates from measurements taken at multiple routers.

1 Introduction

1.1 Background

The increasing speed of network links makes it infeasible to collect complete data on all packets or network flows. This is due to the costs and scale of the resources that would be required to accommodate the data in the measurement infrastructure. These resources are (i) processing cycles at the observation point (OP) which are typically scarce in a router; (ii) transmission bandwidth to a collector; and (iii) storage capacity and processing cycles for querying and analysis at the collector.

These constraints motivate reduction of the data. Of three classical methods—filtering, aggregation and sampling—the first two require knowing the traffic features of interest in advance, whereas only sampling allows the retention of arbitrary detail while at the same time reducing data volumes. Sampling also has the desirable property of being simple to implement and quick to execute, giving it an advantage over recently developed methods for computing compact approximate aggregates such as sketches [14].

Sampling is used extensively in traffic measurement. sFlow [17] sends packet samples directly to a collector. In Trajectory Sampling, each packet is selected either at all points on its path or none, depending on the result of applying a hash function to the packet content [3]. In Sampled NetFlow [1], packets are sampled before the formation of flow statistics, in order to reduce the speed requirements for flow cache lookup. Several methods focus measurements on the small proportion of longer traffic flows that contain a majority of packets. An adaptive packet sampling scheme for keeping flow statistics in routers which includes a binning scheme to keep track of flows of different lengths is proposed in [7]. Sample and Hold [8] samples new flow cache instantiations, so preferentially sampling longer flows. RATE [12] keeps statistics only on those flows which present successive packets to the router, and uses these to infer statistics of the original traffic. Packet sampling methods are currently being standardized in the Packet Sampling (PSAMP) Working Group of the Internet Engineering Task Force [15]. Flow records can themselves be sampled within the measurement infrastructure, either at the collector, or at intermediate staging points. Flow-size dependent sampling schemes have been proposed [4, 5, 6] to avoid the high variance associated with uniform sampling of flows with a heavy tailed length distribution.

1.2 Motivation

Multiple Traffic Measurements. This paper is motivated by the need to combine multiple and possibly overlapping samples of network traffic for estimation of the volumes or rates of matrix elements and other traffic components. By a traffic component we mean a (maximal) set of packets sharing some common property (such as a flow key), present in the network during a specified time frame. Traffic OPs can be different routers, or different interfaces on the same router. Reasons for taking multiple measurements include: (i) all traffic must pass at least one OP; (ii) measurements must be taken at a specified set of OPs; and (iii) network traffic paths must be directly measured.

Sampling and Heterogeneity. Traffic analysis often requires joining the various measurement datasets, while at the same time avoiding multiple counting. Sampling introduces further complexity since quantities defined for the original traffic (e.g. traffic matrix elements) can only be estimated from the samples. Estimation requires both renormalization of traffic volumes in order to take account of sampling, and analysis of the inherent estimator variability introduced through sampling.

Depending on the sampling algorithm used, the proportion of traffic sampled from a given traffic component may depend on (i) the sampling rate (e.g. when sampling uniformly) and/or (ii) the proportion of that component in the underlying traffic (e.g. when taking a fixed number of samples from a traffic population). Spatial heterogeneity in traffic rates and link speeds presents a challenge for estimating traffic volumes, since a traffic component may not be well represented in measurements all points, and sampling rates can differ systematically across the network. For example, the sampling rate at a lightly loaded access link may be higher than at a heavily loaded core router. Changes in background traffic rates (e.g. due to attacks or rerouting) can cause temporal heterogeneity in the proportion of traffic sampled.

Combining Estimates. This paper investigates how best to combine multiple estimates of a given traffic component. Our aim is to minimize the variability of the combined estimate. We do this by taking a weighted average of the component estimates that takes account of their variances. Naturally, this approach requires that the variance of each component is known, or can at least be estimated from the measurements themselves. A major challenge in this approach is that inaccurate estimates of the variance of the components can severely impair the accuracy of the combination. We propose robust solutions that adapt to estimated variances while bounding the impact of their inaccuracies.

What are the advantages of adapting to estimated variances, and combining multiple estimates? Why not simply use the estimate with lowest variance? The point of adaptation is that the lowest variance estimate cannot generally be identified in advance, while combining multiple estimates gains significant reduction in variance.

The component estimators are aggregates of individual measurements. Their variances can be estimated provided the sampling parameters in force at the time of measurement are known. This is possible when sampling parameters are reported together the measurements, e.g., as is done by Cisco Sampled NetFlow [2]. The estimated variance is additive over the measurements. This follows from a subtle but important point: we treat the underlying traffic as a single fixed sample path rather than a statistical process. The only variance is due to sampling, which can be implemented to be independent over each packet or flow record. Consequently, variance estimates can be aggregated along with the estimates themselves, even if the underlying sampling parameters change during the period of aggregation.

We now describe two scenarios in which multiple overlapping traffic measurement datasets are produced, in which our methodology can be usefully applied. We also mention a potential third application, although we do not pursue it in this paper.

1.3 Router Matrix Estimation

Router Measurements and Matrix Elements. Applications such as traffic engineering often entail determining traffic matrices, either between ingress-egress interface pairs of a router, or at finer spatial scales, e.g., at the routing prefix level or subnet level matrices for traffic forwarded through a given ingress-egress interface pair. A common approach to traffic matrix estimation is for routers to transmit reports (e.g. packet samples or NetFlow statistics) to a remote collector, where aggregation into matrix elements (MEs) is performed.

Observation Points and Sampling Within a Router. The choice of OPs within the router can have a great effect on the accuracy of traffic matrices estimated from samples. Consider the following alternatives:

- *Router-level Sampling*: all traffic at the router is treated as a single stream to be sampled. We assume ingress and egress interface can be attributed to the measure traffic, e.g., as reported by NetFlow.

- *Unidirectional Interface-level Sampling:* traffic is sampled independently in one direction (incoming or outgoing) of each interface.

- *Bidirectional Interface-level Sampling:* traffic is sampled independently in both interface directions.

Comparing Sampling at the Observation Points. Accurate estimation of an ME requires sufficiently many flows to be sampled from it. For example, in uniform sampling with probability p, the relative standard deviation for unbiased estimation of the total bytes of n flows behaves roughly as $\sim 1/\sqrt{np}$. We propose two classes of important MEs:

(i) *Large matrix elements:* these form a significant proportion of the total router traffic.

(ii) *Relatively large matrix elements:* these form a significant proportion of the traffic on either or both of their ingress or egress router interfaces. (We use the terms *small* and *relatively small* in an obvious way).

Gravity Model Example. In this case the ME m_{xy} from interface x to interface y is proportional to $M_x^{\text{in}} M_y^{\text{out}}$ where M^{in} and M^{out} denote the interface input and output totals; see [13, 18]. The large MEs m_{xy} are those for which both M_x^{in} and M_y^{out} are large. The relatively large MEs are those for which either M_x^{in} or M_y^{out} (or both) are large.

Router level sampling is good for estimating large MEs, but not those that are only relatively large at the router level. This is because the sampling rate is independent of its ingress and egress interfaces. In the gravity model, router sampling is good for estimating the "large-to-large" MEs, (i.e. those m_{xy} for which both M_x^{in} and M_y^{out} are large) but not good for estimating "large-to-small" and "small-to-large" (and "small-to-small") MEs.

Unidirectional interface-level sampling offers some improvement, since one can use a higher sampling rate on interfaces that carry less traffic. However, unidirectional sampling, say on the ingress direction, will not help in getting sufficient samples from a small interface-to-interface traffic ME whose ingress is on an interface that carries a high volume of traffic. In the gravity model, "large-to-small" (and "small-to-small") MEs would be problematic with ingress sampling.

Only bidirectional interface-level sampling can give a representative sample of small but relatively large MEs. Two different estimates of the MEs could be formed, one by selecting from an ingress interface all samples destined for a given egress interface, and one by selecting from an egress interface all samples from a given input interface. The two estimates are then combined using the method proposed in this paper.

The effectiveness of router or interface level sampling for estimating large or relatively large MEs depends on the sampling rates employed and/or the resources available for storing the samples in each case. If router level *and* interface level sampling are employed, three estimates (from router, ingress and egress sampling) can be combined. In both the three-way and two-way combinations, no prior knowledge is required of sampling parameters or the sizes of the MEs or their sizes relative to the traffic streams from which they are sampled.

Resources and Realization. The total number of samples taken is a direct measure of the memory resources employed. We envisage two realizations in which our analysis is useful. Firstly, for router based resources, the question is how to allocate a given amount of total router memory between router based and interface based sampling. The second realization is for data collection and analysis. Although storage is far cheaper than in the router case, there is still a premium on query execution speed. Record sampling reduces query execution time. The question becomes how many samples of each type (interface or router) should be used by queries.

1.4 Network Matrix Estimation Problem

The second problem that we consider is combining measurements taken at multiple routers across a network. One approach is to measure at all edge interfaces, i.e., access routers and peering points. Except for traffic destined to routers themselves, traffic is sampled at both ingress and egress to the network. Estimating traffic matrices between edges is then analogous to the problem of estimating ingress-egress MEs in a single router from bidirectional interface samples.

Once measurement and packet sampling capabilities become standardized through the PSAMP and Internet Protocol Flow Information eXport (IPFIX) [11] Working Groups of the IETF, measurements could be ubiquitously available across network routers. Each traffic flow would potentially be measured at all routers on its path. With today's path lengths, this might entail up to 30 routers [16]. However, control of the total volume of data traffic may demand that the sampling rate at each OP be quite low; estimates from a single OP may be quite noisy. The problem for analysis is how to combine these noisy estimates to form a reliable one.

1.5 Parallel Samples

Multiple sampling methods may be used to match different applications to the statistical features of the traffic. For example, the distribution of bytes and packet per flow has been found to be heavy-tailed; see [10]. For this reason, sampling flow records with a non-uniform probability that is higher for longer flows leads to more accurate estimation of the total traffic bytes than uniform sampling; see [4]. On the other hand, estimates of the number of flows are more accurate with uniform sampling. When multiple sampling methods are used, it is desirable to exploit all samples generated by both methods if this reduces estimator variance.

1.6 Outline

Section 2 describes the basic model for traffic sampling, then describes a class of minimum variance convex combination estimators. The pathologies that arise when using these with estimated variance are discussed. Section 3 proposes two regularized estimators that avoid these pathologies. Section 4 recapitulates two closely related sample designs for size dependent sampling of flow records, and applies the general form of the regularized estimators from Section 3 in each case. The remainder of the paper is concerned with experimental evaluation of the regularized size-dependent estimators for combining samples of flow records. Section 5 evaluates their performance in the router interface-level traffic matrix estimation problem of Section 1.3, and demonstrates the benefits of including interface-level samples in the combination. Section 6 evaluates performance of the regularized estimators in the network matrix estimation problem of Section 1.4 and shows how they provide a robust combination estimates under wide spatial variation in the underlying sampling rate. We conclude in Section 7.

2 Combining Estimators

2.1 Models for Traffic and Sampling

Consider n traffic flows labelled by $i = 1, 2, \ldots, n$, with byte sizes x_i. We aim to estimate the byte total $X = \sum_{i=1}^{n} x_i$. Each flow i can be sampled at one of m OPs, giving rise to estimators $\widehat{X}_1, \ldots \widehat{X}_m$ of X as follows. Let $p_{ij} > 0$ be the probability that flow i is selected at OP j. In general p_{ij} will be a function of the size x_i, while its dependence on j reflects the possible inhomogeneity of sampling parameters across routers.

Let χ_{ij} be the indicator of selection, i.e., $\chi_{ij} = 1$ when the flow i is selected in measurement j, and 0 otherwise. Then each $\widehat{x}_{ij} = \chi_{ij} x_i / p_{ij}$ is an unbiased estimator of x_i, i.e., $\mathsf{E}[\widehat{x}_{ij}] = x_i$ for all measurements j. Renormalization by p_{ij} compensates for the fact that the flow may not be selected. Clearly $\widehat{X}_j = \sum_{i=1}^{n} \widehat{x}_{ij}$ is an unbiased estimator of X. Note the x_i are considered deterministic quantities; the randomness in the $\widehat{X}_i$ arises only from sampling. We assume that the sampling decisions (the χ_{ij}) for each flow i at each of the m OPs are independent; it follows that the $\widehat{X}_j$ are independent.

2.2 Variance of Combined Estimators

In order to use all the information available concerning X, we form estimators of X that depend jointly on the m estimators $\widehat{X}_1, \ldots, \widehat{X}_m$. We focus on convex combinations of the $\widehat{X}_j$, i.e., estimators of the form

$$\widehat{X} = \sum_{j=1}^{m} \lambda_j \widehat{X}_j, \text{ with } \lambda_j \in [0, 1], \ \sum_{j=1}^{m} \lambda_j = 1. \quad (1)$$

We allow the coefficients λ_j to be random variables than can depend on the $\widehat{x}_{ij}$. This class of models is reasonably amenable to analysis, and the statistical properties of its members are relatively easy to understand.

Each choice of the coefficients $\underline{\lambda} = \{\lambda_j : j = 1, \ldots, m\}$ gives rise to an estimator $\widehat{X}$. Which $\underline{\lambda}$ should be used? To evaluate the statistical properties of the estimators (1), we focus on two properties: bias and variance. We now describe these for several cases of the estimator (1). Let v_j denote the variance $\mathsf{Var}(\widehat{X}_j)$, i.e,

$$v_j = \mathsf{Var}(\widehat{X}_j) = \sum_{i=1}^{n} \mathsf{Var}(\widehat{x}_{ij}) = \sum_{i=1}^{n} \frac{x_{ij}^2 (1 - p_{ij})}{p_{ij}} \quad (2)$$

2.3 Average Combination Estimator

Here $\lambda_j = 1/m$ hence $\widehat{X} = m^{-1} \sum_{j=1}^{m} \widehat{X}_j$. This estimator is unbiased since the λ_j are independent : $\mathsf{E}[\widehat{X}] = \sum_{j=1}^{m} \lambda_j \mathsf{E}[\widehat{X}_j] = X$. It has variance $\mathsf{Var}(\widehat{X}) = m^{-2} \sum_{j=1}^{m} v_j$. This estimator is very simple to compute. However, it suffers from sensitivity of $\mathsf{Var}(\widehat{X})$ to one constituent estimator $\widehat{X}_j$ having large variance v_j, due to. e.g., a small sampling rate. The average estimator is special case of the following class of estimator.

2.4 Independent $\{\lambda_j\}$ and $\{\widehat{X}_j\}$.

When λ_j is independent of $\widehat{X}_j$, $\widehat{X}$ is unbiased, since

$$\mathsf{E}[\widehat{X}] = \mathsf{E}[\mathsf{E}[\widehat{X} | \underline{\lambda}]] = X \mathsf{E}[\sum_{j=1}^{m} \lambda_j] = X \quad (3)$$

Furthermore, elementary algebra shows that

$$\mathsf{Var}(\widehat{X}) = \sum_{j=1}^{m} \mathsf{E}[\lambda_j^2] v_j \quad (4)$$

The RHS of (4) can be rewritten as

$$\sum_{j=1}^{m} \mathsf{E}[\lambda_i^2] v_j = \sum_{j=1}^{m} \mathsf{E}[(\lambda_j - \Lambda_j(\underline{v}))^2] v_j + V_0(\underline{v}) \quad (5)$$

where

$$\Lambda_j(\underline{v}) = \frac{1/v_j}{\sum_{j'=1}^{m} 1/v_{j'}} \quad , \quad V_0(\underline{v}) = 1 / \sum_{j=1}^{m} v_j^{-1} \quad (6)$$

Eq. (5) shows that the variance of $\widehat{X}$ is minimized by minimizing the total mean square error in estimating the Λ_j by λ_j. Then $V_0(\underline{v})$ is the minimum variance that can be attained. The form of Λ_j says that the more reliable estimates, i.e., those with smaller variance, have a greater impact on the final estimator.

2.5 Estimators of Known Variance

For known variances v_j, $\mathsf{Var}(\widehat{X})$ is minimized by

$$\lambda_j = \Lambda_j(\underline{v}) \quad (7)$$

We do not expect the v_i will be known a priori. For general p_{ij} it is necessary to know all x_i in order to determine v_i. However, in many applications, only the sizes x_i of those flows actually selected during sampling will be known. We now mention two special cases in which the variance is at least implicitly known.

2.6 Spatially Homogeneous Sampling

Each flow is sampled with the same probability at each OP, which may differ between flows: $p_{ij} = p_i$ for some p_i and all j. Then the v_i are equal and we take $\lambda_j = \Lambda_j(\underline{v}) = 1/m$. Hence for homogeneous sampling, the average estimator from Section 2.3 is the minimum variance convex combination of the $\widehat{X}_j$.

2.7 Pointwise Uniform Sampling

Flows are sampled uniformly at each OP, although the sampling probability may vary between points: $p_{ij} = q_j$ for some q_j and all i. Then $v_j = (\sum_{i=1}^n x_i^2)u_j$ where $u_j = (1 - q_j)/q_j$. The dependence of each v_j in the $\{x_i\}$ is a common multiplier which cancels out upon taking the minimum variance convex combination $\widehat{X}$ using

$$\lambda_j = \Lambda_j(\underline{v}) = \Lambda_j(\underline{u}) \tag{8}$$

2.8 Using Estimated Variance

When variances are not know a priori, they may sometimes be estimated from the data. For each OP j, and each flow i, the random quantity

$$\widehat{v}_{ij} = \chi_{ij} x_i^2 (1 - p_{ij})/p_{ij}^2 \tag{9}$$

is an unbiased estimator of the variance $v_{ij} = \mathsf{Var}(\widehat{x}_{ij})$ in estimating x_i by $\widehat{x}_{ij}$. Hence

$$\widehat{V}_j = \sum_{i=1}^n \widehat{v}_{ij} \tag{10}$$

is an unbiased estimator of v_j. Put another way, we add an amount $x_i^2(1 - p_{ij})/p_{ij}^2$ to the estimator $\widehat{V}_j$ whenever flow i is selected at observation point j.

Note that $\widehat{V}_j$ and $\widehat{X}_j$ are dependent. This takes us out of the class of estimators with independent $\{\lambda_j\}$ and $\{\widehat{X}_j\}$, and there is no general simple form for the $\mathsf{Var}(\widehat{X})$ analogous to (4). An alternative is to estimate the variance from an independent set of samples at each OP j. This amounts to replacing χ_{ij} by an independent identically distributed sampling indicator $\{\chi'_{ij}\}$ in (9). With this change, we know from Section 2.4 that using

$$\lambda_j = \Lambda_j(\widehat{\underline{V}}) \tag{11}$$

will result in an unbiased estimator $\widehat{X}$ in (1). But the estimator will not in general have minimum possible variance $V_0(\underline{v})$ since λ_j is not necessarily an unbiased estimator of $\Lambda_j(\underline{v})$.

2.9 Some Ad Hoc Approaches

A problem with the foregoing is that an estimated variance $\widehat{V}_j$ could be zero, causing $\Lambda_j(\widehat{\underline{V}})$ to be undefined. On the other hand, the average estimator is susceptible to the effect of high variances. Some ad hoc fixes include:

AH1: Use $\lambda_j = \Lambda_j(\widehat{V})$ on the subset of sample sets j with non-zero estimated variance. If all estimated variances are zero, use the average estimator.

AH2: Use the non-zero estimate of lowest estimated variance. But these estimators still suffer from a potentially far more serious pitfall: the impact of statistical fluctuations in small estimated variances. This is discussed further in Section 2.10.

2.10 Discussion

Absence of Uniformity and Homogeneity. We have seen in Section 2.6 that the average estimator is the minimum variance convex combination only when sampling is homogeneous across OPs. In Section 2.7 we saw that we can form a minimum variance estimator without direct knowledge of estimator variance only when sampling is uniform. In practice, we expect neither of these conditions to hold for network flow measurements.

Firstly, sampling rates are likely to vary according to monitored link speed, and may be dynamically altered in response to changes in traffic load, such as those generated by rerouting or during network attacks. In one proposal, [7], the sampling rate may be routinely changed on short time scales during measurement, while the emerging PSAMP standard is designed to facilitate automated reconfiguration of sampling rates. Secondly, the recognition of the concentration of traffic in heavy flows has led to sampling schemes in which the sampling probability of a flow (either of the packets that constitute it, or the complete flow records), depends on the flow's byte size rather than being uniform; see [4, 5, 6, 8, 12]. Finally, in some sampling schemes, the effective sampling rate for an item is a random quantity that depends on the whole set of items from which it is sampled, and hence varies when different sets are sampled from. Priority sampling is an example; see Section 4.

Pathologies of Small Estimated Variances. Using estimated variances brings serious pitfalls. The most problematic of these is that samples taken with a low sampling rate may have estimate variance close to or even equal to zero. Even if the zero case is excluded in ad hoc manner, e.g. as described in Section 2.9, a small and unreliable sample may spuriously dominate the estimate because its estimated variance happens to be small. Some form of regularization is required in order to alleviate this problem. A secondary issue for independent variance estimation is the requirement to maintain a second set of samples, so doubling resource requirements.

In the next sections we propose a regularization for variance estimation in a recently proposed flow sampling scheme that controls the effect of small estimated variances, even in the dependent case.

3 Regularized Estimators

We propose two convex combination estimators of the type (1) using random coefficients $\{\lambda_j\}$ of the form (11) but regularizing or bounding the variances to control the impact of small estimated variances. Both estimators take the form $\sum_j \lambda_j \widehat{X}_j$ with $\lambda_j = \Lambda_j(\underline{\widehat{U}})$ for some estimated variances $\underline{\widehat{U}}$, while they differ in which $\underline{\widehat{U}}$ is used.

Both estimators are characterized by the set of quantities $\underline{\tau}$, where for each OP j:

$$\tau_j = \max_{i:\, p_{ij}<1} (x_i/p_{ij}) \tag{12}$$

The τ_j may be known a priori from a given functional dependence of p_{ij} on x_i, or it may only be known from the measurements themselves.

3.1 Regularized Variance Estimator

The first estimator ameliorates the impact of small underestimated variances, while still allowing combination to take account of different but well-estimated variances. Note that the estimated variance $\widehat{v}_{ij}$ obeys the bound

$$\widehat{v}_{ij} \leq \chi_{ij}\tau_j^2 \tag{13}$$

This suggests that we can ameliorate the effects of random exclusion of a flow from a sample by adding a small multiple s of τ_j^2 to each variance estimator $\widehat{V}_j$. This represents the scale of uncertainty in variance estimation. The addition has little effect when the estimated variance arises from a large number of samples, but tempers the effect of a small sample for which the variance happens to be small or even zero. With this motivation, the **regularized variance estimator** is $\widehat{X} = \sum_j \lambda_j \widehat{X}_j$ with

$$\lambda_j = \Lambda_j(\underline{\widehat{V}}') \quad \text{where} \quad \widehat{V}_j' = \widehat{V}_j + s\tau_j^2 \tag{14}$$

The corresponding variance estimate for this convex combination is $\widehat{V} = \sum_{j=1}^m \lambda_j^2 \widehat{V}_j$. The special case $s = 0$ is just the estimator from Section 2.8.

3.2 Bounded Variance Estimator

The second estimator uses a similar approach on the actual variance v_{ij}, which obeys the bound:

$$v_{ij} \leq x_i\tau_j \tag{15}$$

If this bound were equality, we would then have $V_j = X\tau_j$, in which case, the minimum variance estimator would be the **bounded variance estimator**, namely, $\widehat{X} = \sum_j \lambda_j \widehat{X}_j$ with $\lambda_j = \Lambda_j(X\underline{\tau}) = \Lambda(\underline{\tau})$. The corresponding variance estimate for this convex combination is $\widehat{V} = \sum_{j=1}^m \lambda_j^2 \widehat{V}_j$. The strength of this approach is that the variance estimate can take account of knowledge of inhomogeneity in the sample rates (as reflected by inhomogeneity in the τ_j) while not being subject to statistical fluctuations in variance estimates.

Uniform and Homogeneous Sampling. Note that uniform and homogeneous sampling fall into this framework already (with equality in (15)), since in both cases the dependence of the variances v_j on the objects x_i to be sampled is a common factor over all OPs j, which is hence eliminated from the coefficients λ_j.

Small Sampling Probabilities. The tightness of the bound (15) depends on the functional form of p_{ij}. One particular case is when sampling probabilities are small. For this case we propose a linear approximation:

$$p_{ij} = x_i/\tau_j + O((x_i/\tau_j)^2) \tag{16}$$

This yields approximate equality in (15), provided all x_i are small compared with τ_j. We give an example of a sample design with this property in Section 4.

3.3 Confidence Intervals

We form approximate conservative confidence intervals for $\widehat{X}$ by applying a regularization of the type (14). Thus the upper and lower confidence intervals are

$$\widehat{X}_{\pm} = \widehat{X} \pm s(\widehat{V} + s\tau^2) \tag{17}$$

where s is the target number of standard deviations away from the mean.

4 Size Dependent Flow Sampling

The remainder of the work in this paper will focus on two closely related schemes for sampling completed flow records. These are **threshold sampling** [4] and **priority sampling** [6]. We briefly recapitulate these now.

4.1 Threshold Sampling

For a threshold $z > 0$, a flow of size x is sampled with probability $p_z(x) = \min\{1, x/z\}$. Thus flows of size $x \geq z$ are always sampled, while flows of size $x < z$ are sampled with probability proportional to their size. This alleviates the problem of uniform sampling, that byte estimation can have enormous variance due to random selection or omission of large flows. In threshold sampling, all flows of size at least z are always selected.

Starting with a set of flows with sizes $\{x_i\}$ as before, we form an unbiased estimator $\widehat{X}$ of $X = \sum_{i=1}^n x_i$ using the selection probabilities $p_i = p_z(x_i)$. (In this section we suppress the index j of the OP). The estimator of X from a single OP takes the form $\widehat{X}$ takes the specific form

$$\widehat{X} = \sum_{i=1}^n \chi_i x_i/p_z(x_i) = \sum_{i=1}^n \chi_i \max\{x_i, z\} \tag{18}$$

Threshold sampling is optimal in the sense that it minimizes the cost $C_z = \mathsf{Var}(\widehat{X}) + z^2 N$ where $N = \sum_{i=1}^n p_i$ is the expected number of samples taken. This cost expresses the balance between the opposing goals of reducing the number of samples taken, and reducing the uncertainty in estimating X. The value of z determines the relative importance attached to these goals.

Applying the general formula (2), the variance of the estimate $\widehat{X}$ from a single OP is

$$\mathsf{Var}(\widehat{X}) = \sum_{i=1}^{n} x_i \max\{z - x_i, 0\} \tag{19}$$

which has unbiased estimator

$$\widehat{V} = \sum_{i=1}^{n} \chi_i z \max\{z - x_i, 0\} \tag{20}$$

In threshold sampling, inhomogeneity across OPs arises through inhomogeneity of the threshold z.

4.2 Priority Sampling

Priority sampling provides a way to randomly select exactly k of the n flows, weighted by flow bytes, and then form an unbiased estimator of the total bytes X. The algorithm is as follows. For each flow i, we generate a random number α_i uniformly distributed in $(0, 1]$, and construct its *priorities* $\widehat{z}_i = x_i/\alpha_i$. We select the k flows of highest priority. Let $\widehat{z}'$ denote the $(k+1)^{\text{st}}$ highest priority. At a single OP, we for the estimate

$$\widehat{X} = \sum_{i-1}^{n} \chi_i \max\{x_i, \widehat{z}'\} \tag{21}$$

of the total bytes X. Here χ_i is the indicator that flow i is amongst the k flows selected. $\widehat{X}$ is unbiased; see [6].

For priority sampling, the variance of $\widehat{X}$ takes a similar form to that of threshold sampling:

$$\mathsf{Var}(\widehat{X}) = \sum_{i=1}^{n} x_i \mathsf{E}[\max\{\widehat{z}' - x_i, 0\}] \tag{22}$$

which has unbiased estimator

$$\widehat{V} = \sum_{i=1}^{n} \chi_i \widehat{z}' \max\{\widehat{z}' - x_i, 0\} \tag{23}$$

Although sampling of flows is dependent, it turns out that the unbiased estimates $\widehat{x}_i = \chi_i \max\{\widehat{z}, x_i\}$ of the bytes of different flows have zero covariance.

In priority sampling, inhomogeneity between observation points arises not only through inhomogeneity of the number of flows k selected, but also through the background traffic. Typically we want to estimate the total bytes not of all sampled flows, but only of a selection of them that share some property of interest, e.g., a specific source and destination. The probability that a given interesting flow will be amongst the k flows selected, depends also on the sizes of all flows in the background traffic, which generally varies between different OPs. Threshold sampling is independent between flows.

4.3 Threshold and Priority Compared

The estimator (21) appears quite similar to that for threshold sampling (18), except that the role of the threshold z is played by the random quantity $\widehat{z}'$. In fact, the relationship is deeper: one can show that, conditioned on the threshold $\widehat{z}'$, the selection probabilities for each flow minimize a cost analogous to C_z.

For applications, we see that threshold sampling is well suited to streaming applications when buffer space is expensive (e.g., at a router) since each object is sampled independently. Priority sampling is able to constrain the number of samples taken, at the cost of maintaining a buffer of k candidate samples during selection. It is well suited to applications where buffering is less expensive (e.g., in a data aggregator or database)

4.4 Regularized Variance Estimators

Threshold and priority sampling both give rise to regularized estimators as described in Section 3. Consider first threshold sampling and let z_j be the sampling threshold in force at OP j. Then the quantity τ_j in (12) is just z_j. Moreover, p_{ij} is approximately linear in x_i, the sense of (16), and hence the bounded variance estimator is expected to perform reasonably for flows whose size x_i are small compared with the z_j. For priority sampling, we use the random thresholds z'_j in place of the z_j. Although this introduces additional variability; in practice priority approximates threshold sampling closely for large number of samples. In the next sections we show this heuristic performs well in experiments.

5 Experiments: Router Matrix

This section applies our method to traffic measurement at routers. As discussed in Section 1.3, while router level sampling captures large MEs accurately, interface level sampling offers the opportunity to accurately sample not just the relatively large ones MEs, i.e., the largest amongst those seen at each interface. This is particularly important for a method such as priority sampling where, in order to provide a hard constraint on the use of measurement resources, only a fixed number of samples are taken in a given time period, There is a trade-off: if all resources were deployed for interface sampling, then not all larger flows on some heavily used interfaces might be sampled.

		1	2	3	4	5	6	7	8
		0.0004	0.04	0.1	0.004	0.03	0.8	0.02	0
1	0	0	0	0	0	0	0	0	0
2	0.5	8e-05	0	0.0007	0	0	0.5	0.0001	0
3	0.01	7e-05	0.0002	0	0	0.001	0.01	0.0004	0
4	0	0	0	0	0	0	0	0	0
5	0.2	2e-05	0	0.05	0.003	3e-05	0.1	0.006	0
6	0.3	0.0002	0.04	0.08	0.001	0.02	0.2	0.01	0
7	0.01	2e-05	0.003	0.0004	5e-06	0.006	0.0007	3e-05	0
8	1e-06	1e-06	0	0	0	0	0	0	0

Table 1: Router matrix elements for CAMPUS, with row and column sums, normalized by total bytes

This motivates using a combined estimator. In this application we explicitly want to take account of estimated variance, so we use the regularized variance estimator of Section 3. In experiments using real flow data taken at two routers, we find that:
(i) For a given total number of samples, the regularized estimator is more accurate than its individual consistent estimators or averages thereof.
(ii) The regularized estimator is more accurate than the ad hoc estimator AH1 when estimation error is large.

5.1 Router Data and Traffic Matrices

The data from this experiment comprised sampled NetFlow records gathered from two routers in a major ISP network. These record the total bytes of the sampled flow packets, and the router input and output interfaces traversed by the flow. Thus, it is possible to map each flow onto the appropriate router to router traffic matrix.

The first dataset, CAMPUS comprises 16,259,841 NetFlow records collected from a backbone router in a corporate intranet during 24 hour period. The active flow timeout was 30 minutes. The maximum size was 3.94 GB and average size 20.4 kB. The router had 8 interfaces. Table 1 shows the interface MEs for a 10 minute period, normalized by total bytes. Note the non-zero MEs range over six orders of magnitude.

The second dataset, DISTRIBUTION comprises 1,765,477 NetFlow records collected during 1 hour from a distribution router in an ISP network. The active flow timeout was 1 minute, with maximum flow size 3.97 MB and average 1.4 kB. The router had 236 interfaces (and subinterfaces), whose line rates ranged from 622 MBps (OC-12) down to 1.5 Mbps (T1). Only 1971 MEs are non-zero. We represent these in Figure 1, where the interfaces have been sorted in decreasing order of total input and output bytes in the 1 hour period. The distribution of traffic per interface is highly skewed: the busiest interface carries 46% of the bytes, while the 10 busiest together carry 94%.

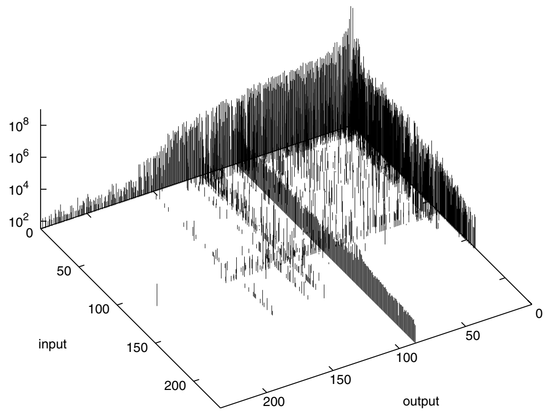


Figure 1: Matrix Elements of Dataset DISTRIBUTION. Interfaces are ordered by total bytes

5.2 Notation for Estimators

`input` and `output` denote the byte estimators derived input and output interface samples respectively, while `router` denote the estimator derived from all flows through the router, undifferentiated by interface. `average` $_{\mathrm{i,o,r}}$ averages `input`, `output` and `router`, while `average` $_{\mathrm{i,o}}$ averages only `input` and `output`. `adhoc` $_{\mathrm{i,o,r}}$ combines the estimators `input`, `output` and `router` as described in AH1 of Section 2.9, while `regular` $_{\mathrm{i,o,r}}$ is the corresponding regularized variance estimator from Section 3. `bounded` is the bounded variance estimator. In priority sampling, `regular` $_{\mathrm{i,o,r}}(k_{\mathrm{i}}, k_{\mathrm{o}}, k_{\mathrm{r}})$ denotes the regularized estimator in which k_{i} and k_{o} priority samples were taken and each input and output interface respectively, and k_{r} were taken at the router level.

A Sample Path Comparison. We compare the performance of the various estimators on several of the CAMPUS MEs from Table 1, as a function of the number of priority

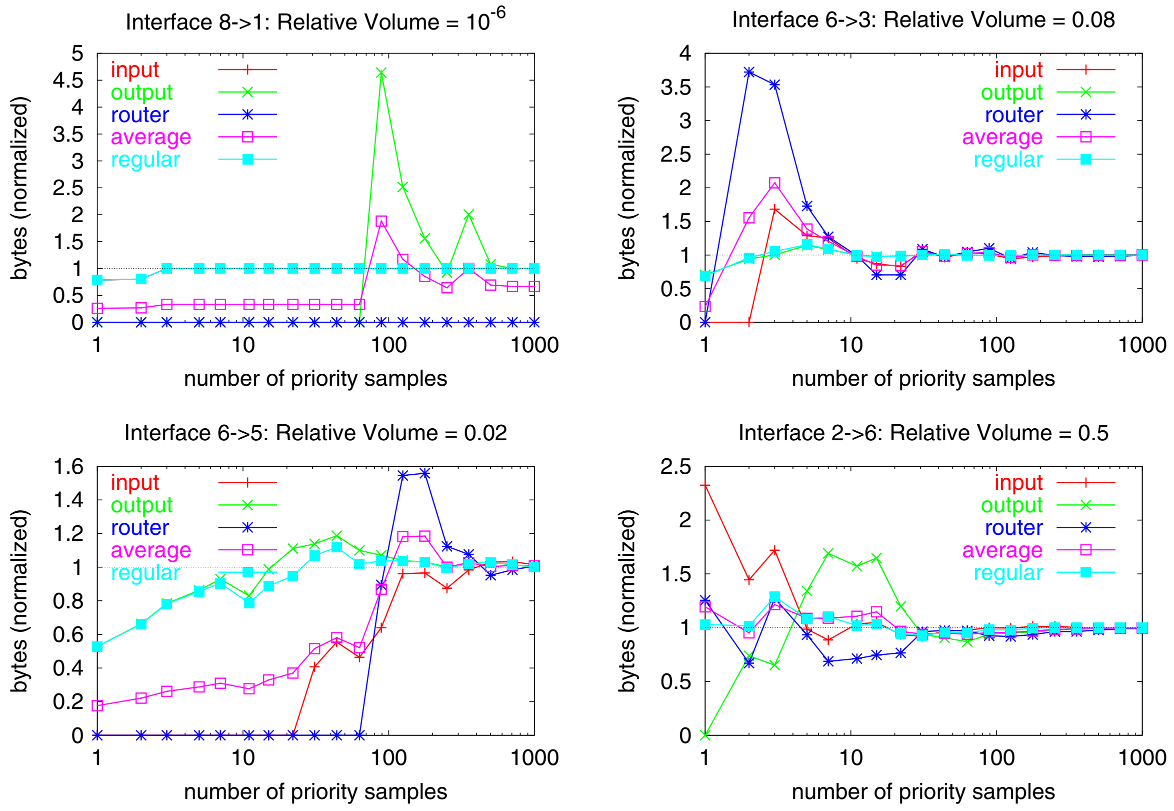


Figure 2: Estimator Comparison: `input`, `output`, `router`, `average` $_{\rm i,o,r}$ and `regular` $_{\rm i,o,r}$, for 4 matrix elements from Table 1 representing various relative volumes of the total bytes.

samples k per interface direction. The estimated MEs (normalized through division by the true value) are displayed in Figure 2 for k roughly log-uniformly distributed between 1 and 1000. Perfect estimation is represented by the value 1. In this evaluation we selected all flows contributing to a given ME, then progressively accumulated the required numbers k of samples from the selection. For this reason, the variation with k is relatively smooth.

There are $N = 8$ interfaces. Each of the single estimators was configured using the same number of sample slots, i.e., `input`(k), `output`(k) and `router`$(2Nk)$. We compare these first; see Figure 2. For the smaller MEs (8→1, 6→3 and 6→5), `input` and `output` are noticeably more accurate than `router`: the relatively large MEs are better sampled at the interface level than at the router level. `average` $_{\rm i,o,r}(k, k, 2Nk)$ performs poorly because of the contribution of `router`, and also because it driven down by the zero estimation from `input` and `output` when the number of samples k is small; see, e.g., the $8 \rightarrow 1$ ME. Only for a large ME (2→6, constituting about half the traffic in the router) does `router` accuracy exceed the worst of the interface methods. Consequently, the accuracy of `average` $_{\rm i,o,r}$ is better in this case too.

When there are noticeable differences between the three single estimators, `regular` $_{\rm i,o,r}(k, k, 2Nk)$ roughly follows the most accurate one. In the $2 \rightarrow 6$ ME, `regular` $_{\rm i,o,r}$ follows `input` most closely while in the $6 \rightarrow 3$ and $6 \rightarrow 5$ MEs, it follows `output`.

5.3 Confidence Intervals

Recall that each estimation method produces and estimate of the variance of the ME estimator. This was used to form upper and lower confidence intervals in Section 3.3. Figure 3 shows upper and lower confidence limits for estimating the MEs of CAMPUS using the same router interfaces as in Figure 2. These use (17) with standard deviation parameter $s = 2$.

8→1 is a special case. `input` has no estimated error when $k \geq 2$. As can be seen from Table 1, 8→1 is the only ME with ingress at interface 8. It comprises 2 flows, so the estimated variance and sampling threshold are 0 for $k \geq 2$. The other methods perform poorly (their confidence bounds are off the chart), since neither `output` nor `router` samples this very small flow.

`regular` $_{\rm i,o,r}$ displays the best overall performance in Figure 2, i.e., it tends to have the smallest divergence from the true value. Figure 3 show that the estimated estimator variance tends to be the smallest too, giving narrower confidence intervals than the other methods.

Estimator Accuracy for Fixed Resources. Now we perform a more detailed comparison of the estimators with the DISTRIBUTION dataset, using constant total sampling slots across comparisons. The router has $N = 236$ interfaces, each bidirectional. For a given number k of sampling slots per interface direction, we compare `router`$(4Nk)$, `input`$(4k)$, `output`$(4k)$, `average`$_{\mathrm{i,o,r}}(k, k, 2Nk)$, `average`$_{\mathrm{i,o}}(2k, 2k)$, `adhoc`$_{\mathrm{i,o,r}}(k, k, 2Nk)$ and `regular`$_{\mathrm{i,o,r}}(k, k, 2Nk)$.

For k values of 16 and 128, and each estimation method, we sorted the relative errors for each ME in increasing order, and plotted them as a function of rank in the left hand column of Figure 4. (The average flow sampling rates are approximately 1 in 234 for $k = 16$ and 1 in 30 for $k = 128$). The curves have the following qualitative features. Moving from left to right, the first feature, present only in some cases, is when the curves start only at some positive rank, indicating all MEs up to that rank have been estimated either with error smaller than the resolution 10^{-5}. The second feature is a curved portion of relative errors smaller than 1. The third feature is a flat portion of relative errors, taking the value 1 for the individual, `adhoc`$_{\mathrm{i,o,r}}$ and `regular`$_{\mathrm{i,o,r}}$ methods, and $1/2$ and $1/3$ for `average`$_{\mathrm{i,o}}$ and `average`$_{\mathrm{i,o,r}}$ respectively. This happens when a ME has no flows sampled by one of the individual estimators. The final feature at the right hand side are points with relative errors $\varepsilon > 1$, indicating MEs that have been overestimated by a factor $\varepsilon + 1$. We make the following observations:
(i) Interface sampling (`input` and `output`) and `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ are uniformly more accurate that `average`$_{\mathrm{i,o,r}}$ or `router`.
(ii) Interface sampling performs better than `adhoc`$_{\mathrm{i,o,r}}$ or `regular`$_{\mathrm{i,o,r}}$ when errors are small. When an ME is very well estimated on a given interface, *any* level information from another interface makes the estimate worse. But when the best interface has a large estimation error, additional information can help reduce it: `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ become more accurate.
(iii) The average-based methods perform poorly; we have argued that they are hobbled by the worst performing component. For example, `average`$_{\mathrm{i,o}}$ performs worse than `input` and `output` since typically only one of these methods accurate for a relatively large ME.
(iv) `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ have similar performance, but when there are larger errors, they are worse on average for `adhoc`$_{\mathrm{i,o,r}}$.
(v) As expected, estimation accuracy increases with the number of samples k, although `average`$_{\mathrm{i,o}}$ and `average`$_{\mathrm{i,o,r}}$ are less responsive.

Although these graphs show that `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ are more accurate than other estimators, is it not immediately evident that this is due to the plausible reasons stated earlier, namely, the more accurate inference of relatively larger flows on smaller interfaces. Also it is not clear the extent to which interface sampling can produce sufficiently accurate estimates at reasonable sampling rates. For example, for k=128 (roughly 1 in 30 sampling of flow records on average) about 25% of the MEs have relative errors 1 or greater. We need to understand which MEs are inaccurately estimated.

To better make this attribution we calculate a scaled version of a MEs as follows. Let Q denote the set of interfaces, and let m_{xy} denote the generic ME from interface x to interface y. Let M^{in} and M^{out} denote the interface input and output totals, so that $M^{\mathrm{in}}_x = \sum_{y \in Q} m_{xy}$ and $M^{\mathrm{out}}_y = \sum_{x \in Q} m_{xy}$. If e_{yx} is the relative error in estimating m_{xy} then we write the scaled version as

$$e'_{xy} = e_{xy} \max\{m_{xy}/M^{\mathrm{in}}_x, m_{xy}/M^{\mathrm{out}}_y\} \qquad (24)$$

Here m_{xy}/M^{in}_x and $m_{xy}/M^{\mathrm{out}}_y$ are the fractions of the total traffic that m_{xy} constitutes on it input and output interfaces. Heuristically, e'_{xy} deemphasizes errors in estimating relatively small MEs.

We plot the corresponding ordered values of the errors e'_{xy} in the right hand column of Figure 4. Note:
(i) `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ are uniformly more accurate than other methods, except for low sampling rates and low estimation errors, in which case they perform about the same as the best of the other methods;
(ii) the accuracy advantage of `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ is more pronounced at larger sampling rates;
(iii) `regular`$_{\mathrm{i,o,r}}$ and `adhoc`$_{\mathrm{i,o,r}}$ display neither the third nor fourth features described above, i.e., no flat portion or errors greater than 1. This indicates that these methods are successful in avoiding larger estimation errors for the relatively large MEs, while for the other methods some noticeable fraction of the relatively large MEs are badly estimated.

We can also get a picture of the relative performance of the methods by looking at the larger estimation errors of the whole traffic matrix. As examples, we show in Figure 5 unscaled relative errors for $k = 128$ samples per interface direction, for `average`$_{\mathrm{i,o}}$ and `regular`$_{\mathrm{i,o,r}}$. Errors have been truncated at 10 in order to retain detail for smaller errors. Observe:
(i) `average`$_{\mathrm{i,o}}$ is poor at estimating many MEs through the largest interface (labeled 1) since smaller MEs are poorly sampled at that interface. `regular`$_{\mathrm{i,o,r}}$ performs better because it uses primarily the estimates gathered at the other interface traversed by these MEs.
(ii) `regular`$_{\mathrm{i,o,r}}$ has a smaller number of large relative errors than `average`$_{\mathrm{i,o}}$.

In order to get a broader statistical picture we repeated the experiments reported in Figure 4 100 times, varying the

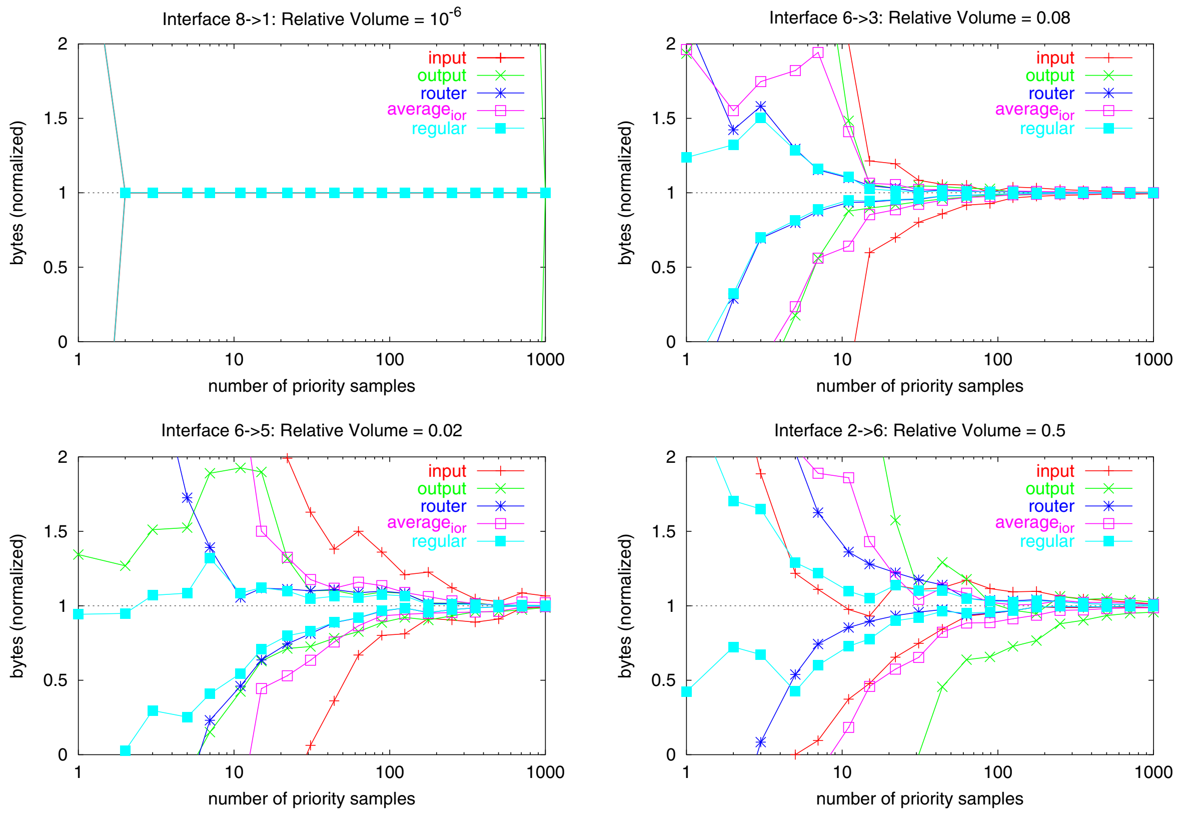


Figure 3: Comparing Confidence Intervals by method, for 4 matrix elements from Table 1

seed for the pseudorandom number generator that governs random selection in each repetition. The ranked root mean square (RMS) of the relative errors shows broadly the same form as Figure 4, but with smoother curves due to averaging over many experiments.

6 Experiments: Network Matrix

In this section we shift the focus to combining a large number of estimates of a given traffic component. Each estimate may individually be of low quality; the problem is to combine them into a more reliable estimate. As mentioned in Section 1.4, this problem is motivated by a scenario in which routers or other network elements ubiquitously report traffic measurements. A traffic component can generate multiple measurements as it transits the network.

A challenge in combining estimates is that they may be formed from sample sets drawn with heterogeneous sampling rates and hence the estimates themselves may have differing and unpredictable accuracy, as described in Section 2.10. For this reason, the approach of Section 3 is appealing, since estimation requires no prior knowledge of sampling rates; it only assumes reporting of the sampling rate in force when the sample was taken.

6.1 Experimental Setup

We wished to evaluate the combined estimator from independent samples of a traffic stream from multiple points. Since we do not have traces taken from multiple locations, we used instead multiple independent samples sets of the CAMPUS flow trace, each set representing the measurements that would be taken from a single OP. We took 30 sample sets in all, corresponding to the current maximum typical hop counts in internet paths [16].

The experiments used threshold sampling, rather than priority sampling, since this would have required the additional complexity of simulating background traffic for each observation point. Apart from packet loss or the possible effects of routing changes, the multiple independent samples correspond with those obtained sampling the same traffic stream at multiple points in the network.

Our evaluations used multiple experiments, each of which represented sampling of a different set of flows in the network. The flow sizes were taken from successive portions of the CAMPUS trace (wrapping around if necessary), changing the seed pseudorandom number generator used for sampling in each experiment. The estimates based on each set of independent samples were combined using

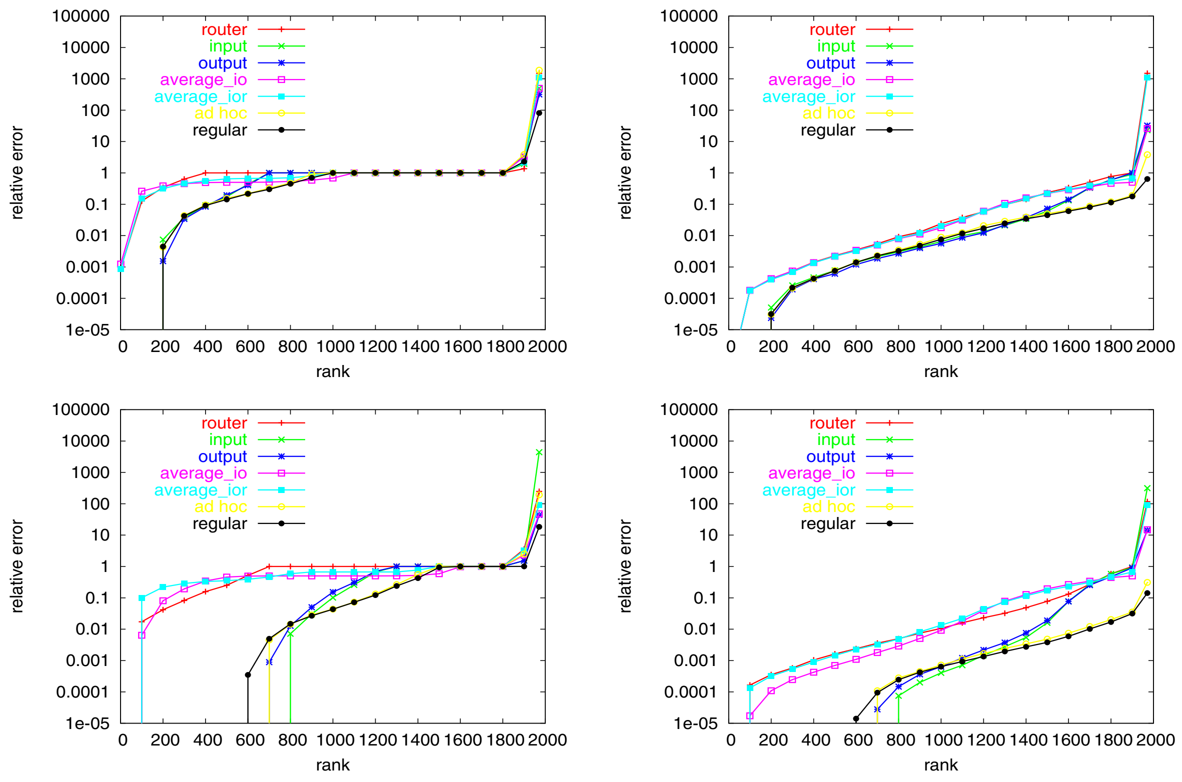


Figure 4: Relative Errors of Matrix Elements for Different Estimators, Ranked by Size. Left: raw relative errors. Right: scaled relative errors. Top: 16 slots per interface. Bottom: 128 slots per interface.

the following methods: `average`, `adhoc`, `bounded` and `regular`. As a performance metric for each method, we computed the root mean square (RMS) relative estimation error over 100 experiments.

6.2 Homogeneous Sampling Thresholds

As a baseline we used a uniform sampling threshold at all OPs. In this case that `bounded` reduces to `average`. In 7 separate experiments we use a sampling threshold of 10^i Bytes for $i = 3, \dots, 9$. This covers roughly the range of flow sizes in the CAMPUS dataset, and hence includes the range of z values that would likely be configured if flow sizes generally conformed to the statistics of CAMPUS. The corresponding sampling rate (i.e. the average proportion of flows that would be selected) with threshold z is $\pi(z) = \sum_i \min\{1, x_i/z\}/N$ where $\{x_i : i = 1, \dots, N\}$ are the sizes of the N flows in the set. For this dataset $\pi(z)$ ranged from $\pi(10^3) = 0.018$ to $\pi(10^9) = 1.9 \times 10^{-5}$.

We show a typical single path of the byte estimate (normalized by the actual value) for a single experiment in Figure 6. This was for 10,000 flows sampled with threshold 10MB at 100 sites. There were typically a handful of flows sampled at each OP. The `bounded` estimate relaxes slowly towards the true value. `regular` also follows at a similar rate, but displays some bias. `adhoc` displays systematic bias beyond 30 combinations. The bias strikingly shows the need for robust estimation methods of the type proposed in this paper.

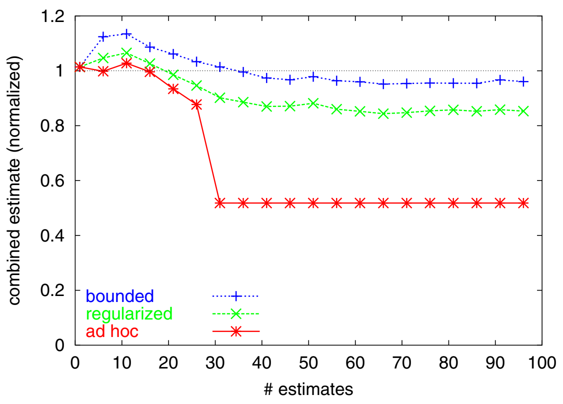
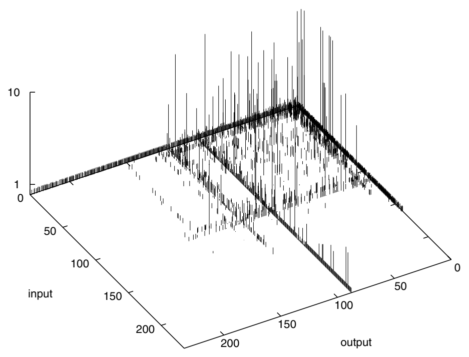


Figure 6: Combined estimators acting cumulatively over 100 independent estimates.

Summary RMS error statistics over multiple experiment are shown in Tables 2 and 3. Here we vary the number of



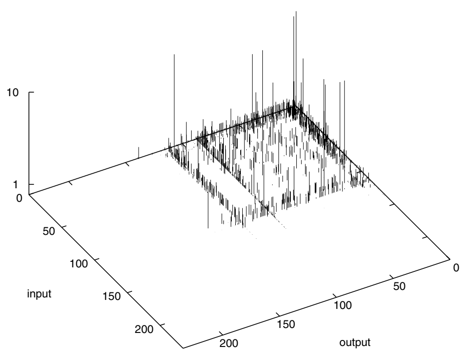


Figure 5: Matrix of relative errors. $k = 128$ samples per interface direction. Left: `average` $_{\mathrm{i,o}}$. Right: `regular` $_{\mathrm{i,o,r}}$.

threshold	`adhoc`	`bounded`	`regular`
10^3	0.0017	0.0016	0.0017
10^4	0.0121	0.0066	0.0117
10^5	0.1297	0.0353	0.0883
10^6	0.4787	0.1293	0.3267
10^7	8.080	0.515	0.527
10^8	46.10	1.464	0.923
10^9	108.7	3.581	1.926

Table 2: Homogeneous Sampling. RMS relative error; 1000 flows, 30 sites

threshold	`adhoc`	`bounded`	`regular`
10^3	0.00002	0.00002	0.00002
10^4	0.00012	0.00012	0.00012
10^5	0.00064	0.00063	0.00064
10^6	0.00340	0.00321	0.00339
10^7	0.01505	0.01110	0.01469
10^8	0.16664	0.05400	0.11781
10^9	0.78997	0.17387	0.37870

Table 3: Homogeneous Sampling. RMS relative error; 100,000 flows, 30 sites

flows in the underlying population (1000 or 100,000) for 30 measurement sites. (Results for 10 measurement sites are not displayed due to space constraints). `bounded` has somewhat better performance than `regular` and significantly better performance than `adhoc`. The differences are generally more pronounced for 30 sites than for 10, i.e., `bounded` is able to take the greatest advantage (in accuracy) of the additional information. On the basis of examination of a number of individual experiments of the type reported in Figure 6, this appears to be due to lower bias in `bounded`.

6.3 Heterogeneous Sampling Thresholds

To model heterogeneous sampling rates we used 30 sampling thresholds in a geometric progression from 100kB to 100MB, corresponding to average sampling rates of from 0.016 to 8.9×10^{-5}. This range of z values was chosen to encompass what we expect would be a range of likely operational sampling rates, these being quite small in order to achieve significant reduction in the volume of flow records through sampling.

We arranged the thresholds in increasing order $10^5 B = z_1 < \ldots < z_i < \ldots < z_{30} = 10^8 B$, and for each m computed the various combined estimators formed from the m individual estimators obtained from samples drawn using the m lowest thresholds $\{z_i : i = 1, \ldots, m\}$. The performance on traffic streams comprising 10,000 flows is shown in Figure 7. Qualitatively similar results were found with 1,000 and 100,000 flows.

The RMS error of `average` initially decreases with path length as it combines the estimators of lower variance (higher sampling rate). But it eventually increases as it mixes in estimators or higher variance (lower sampling rate). RMS errors for `bounded` and `regular` are essentially decreasing with path length, with `bounded` having slightly better accuracy. The minimum RMS errors (over all path lengths) of the three methods a roughly the same. Could `average` be adapted to select and include only those estimates with low variance? This would require an additional decision of which estimates to include, and the best trade-off between accuracy and path length is not known a priori. On the other hand, `bounded` and `regular` can be used with *all available data*, even with constituent estimates of high variance, without apparent degradation of accuracy.

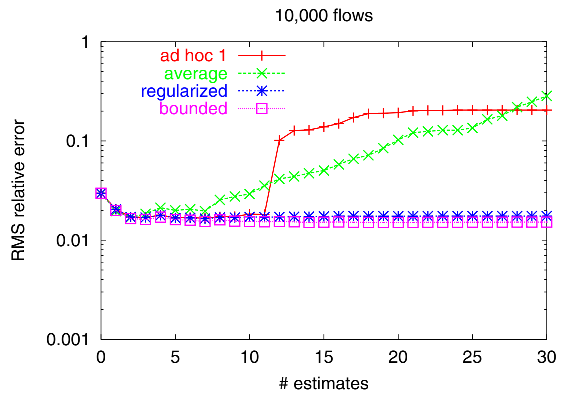


Figure 7: Heterogeneous Sampling Rates. RMS relative errors for `adhoc`, `average`, `regular` and `bounded`, as a function of number of estimates combined.

7 Conclusions

This paper combines multiple estimators of traffic volumes formed from independent samples of network traffic. If the variance of each constituent is known, a minimum variance convex combination can be formed. But spatial and temporal variability of sampling parameters mean that variance is best estimated from the measurements themselves. The convex combination suffers from pathologies if used naively with estimated variances. This paper was devoted to finding remedies to these pathologies.

We propose two regularized estimators that avoid the pathologies of variance estimation. The regularized variance estimator adds a contribution to estimated variance representing the likely sampling error, and hence ameliorates the pathologies of estimating small variances while at the same time allowing more reliable estimates to be balanced in the convex combination estimator. The bounded variance estimator employs an upper bound to the variance which avoids estimation pathologies when sampling probabilities are very small.

We applied our methods to two networking estimation problems: estimating interface level traffic matrices in routers, and combining estimates from ubiquitous measurements across a network. Experiments with real flow data showed that the methods exhibit: (i) reduction in estimator variance, compared with individual measurements; (ii) reduction in bias and estimator variance, compared with averaging or ad hoc combination methods; and (iii) application across a wide range of inhomogeneous sampling parameters, without preselecting data for accuracy. Although our experiments focused on sampling flow records, the basic method can be used to combine estimates derived from a variety of sampling techniques, including, for example, combining mixed estimates formed from uniform and non-uniform sampling of the same population.

Further work in progress examines the properties of combined estimators at an analytical level, and yields a deeper understanding of their statistical behavior beyond the mean and variance.

References

[1] Random Sampled NetFlow. Cisco Systems. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123newft/123t/123t_2/nfstatsa.pdf`

[2] Cisco IOS NetFlow Version 9 Flow-Record Format. `http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/tflow_wp.htm#wp1002063`

[3] N. G. Duffield and M. Grossglauser, "Trajectory Sampling for Direct Traffic Observation", *IEEE/ACM Transactions on Networking*, vol. 9, pp. 280-292, 2001.

[4] N.G. Duffield, C. Lund, M. Thorup, "Charging from sampled network usage," ACM SIGCOMM Internet Measurement Workshop 2001, San Francisco, CA, November 1-2, 2001.

[5] N.G. Duffield, C. Lund, M. Thorup, "Learn more, sample less: control of volume and variance in network measurements", IEEE Trans. of Information Theory, vol. 51, pp. 1756-1775, 2005.

[6] N.G. Duffield, C. Lund, M. Thorup, "Flow Sampling Under Hard Resource Constraints", ACM SIGMETRICS 2004.

[7] C. Estan, K. Keys, D. Moore, G. Varghese, "Building a Better NetFlow", in Proc ACM SIGCOMM 04, Portland, OR

[8] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting", Proc SIGCOMM 2002, Pittsburgh, PA, August 19–23, 2002.

[9] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, F. True, "Deriving traffic demands for operational IP networks: methodology and experience", IEEE/ACM Trans. Netw. vol. 9, no. 3, pp. 265–280, 2001.

[10] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient Policies for Carrying Web Traffic over Flow-Switched Networks," *IEEE/ACM Transactions on Networking*, vol. 6, no.6, pp. 673–685, December 1998.

[11] "Internet Protocol Flow Information Export" (IPFIX). IETF Working Group. `http://net.doit.wisc.edu/ipfix/`

[12] M. Kodialam, T. V. Lakshman, S. Mohanty, "Runs bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation", in Proc. IEEE Infocom 2004, Hong Kong, March 7–11, 2004.

[13] J. Kowalski and B. Warfield, "Modeling traffic demand between nodes in a telecommunications network," in *ATNAC'95*, 1995.

[14] S. Muthukrishnan. "Data Stream Algorithms". Available at http://www.cs.rutgers.edu/∼muthu, 2004

[15] "Packet Sampling" (PSAMP). IETF Working Group Charter. `http://www.ietf.org/html.charters/psamp-charter.html`

[16] "Packet Wingspan Distribution", NLANR. See `http://www.nlanr.net/NA/Learn/wingspan.html`

[17] P. Phaal, S. Panchen, N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", RFC 3176, September 2001

[18] Y. Zhang, M.Roughan, N.G. Duffield, A.Greenberg, "Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads", Proceedings ACM Sigmetrics 2003.

The Power of Slicing in Internet Flow Measurement

Ramana Rao Kompella
University of California, San Diego
ramana@cs.ucsd.edu

Cristian Estan
University of Wisconsin-Madison
estan@cs.wisc.edu

Abstract– Network service providers use high speed flow measurement solutions in routers to track dominant applications, compute traffic matrices and to perform other such operational tasks. These solutions typically need to operate within the constraints of the three precious router resources – CPU, memory and bandwidth. Cisco's NetFlow, a widely deployed flow measurement solution, uses a configurable static sampling rate to control these resources. In this paper, we propose Flow Slices, a solution inspired from previous enhancements to NetFlow such as Smart Sampling [8], Adaptive NetFlow (ANF) [10]. Flow Slices, in contrast to NetFlow, controls the three resource bottlenecks at the router using separate "tuning knobs"; it uses packet sampling to control CPU usage, flow sampling to control memory usage and finally multi-factor smart sampling to control reporting bandwidth. The resulting solution has smaller resource requirements than current proposals (up to 80% less memory usage than ANF), enables more accurate traffic analysis results (up to 10% less error than ANF) and balances better the error in estimates of byte, packet and flow counts (flow count estimates up to 8 times more accurate than after Smart Sampling). We provide theoretical analyses of the unbiasedness and variances of the estimators based on Flow Slices and experimental comparisons with other flow measurement solutions such as ANF.

1 Introduction

The role of traffic measurement in operating large scale IP networks requires little or no introduction. Traffic measurement allows network operators to make informed decisions about provisioning and extending their networks, and it helps solve many operational problems. Specialized devices operating on relatively low traffic links can perform complex security analyses that reveal malicious activities [18, 20], monitor complex performance metrics [6], or simply capture packet (header) traces with accurate timestamps [7] to be analyzed offline. Much simpler solutions such as SNMP counters [16] are deployed on even the highest speed links, but they only give measurements of the total volume of the traffic. Flow level measurement at routers [2, 3] offers a good compromise between scalability and the complexity of the traffic analyses supported since it can offer details about the composition of the traffic mix.

In this paper, we propose a new flow measurement solution: *Flow Slices*. The contributions of this paper are both practical and theoretical and we summarize the most important ones here.

- Flow Slices has separate parameters controlling the three possible bottlenecks at the router: processing load, memory, and reporting bandwidth. This separation allows the solution to be applicable in a wide variety of scenarios with different resource constraints.
- The flow slicing algorithm at the core of this solution provides more accurate results than packet sampling using the same amount of memory. Moreover, it enables new measures of traffic such as estimates for the number of active flows. Note: we use Flow Slices to refer to the the complete flow measurement solution proposed in this paper and flow slicing to refer to the algorithm at the core of the solution.
- Flow Slices separates sampling rate adaptation from binning. Adaptive NetFlow uses more router memory and measurement bandwidth because its flow records are active for fixed time intervals (bins). Adaptive sampling rates give Flow Slices the robustness of Adaptive NetFlow without the overheads of binning. See Table 1 for a comparison of various flow measurement solutions.
- We propose multi-factor smart sampling that takes into account multiple factors such as byte counts, packet counts, and the existence of SYN flags in the flow records to determine the sampling probability for individual flow records. For comparable configurations, this decreases significantly the variance in estimates of the number of flow arrivals while increasing only slightly the variance for byte counts when compared to Smart Sampling.
- Optional binned measurement allows us to eliminate binning error in the analysis phase, while still maintaining the memory and reporting bandwidth overheads below those of Adaptive NetFlow.
- We propose novel estimators $\widehat{b}$, $\widehat{f}$, $\widehat{A}^{(1)}$, and $\widehat{A}^{(2)}$ for various measures of traffic. See Section 4 for a discussion of these and other estimators.

Before we explain Flow Slices, we briefly review some of the previous work in Internet flow measurement.

2 Related work

NetFlow [17], first implemented in Cisco routers, is the most widely used flow measurement solution today. Routers maintain flow records collecting various bits of information. Flows are identified by fields present in the header of every packet: source and destination IP address, protocol, source and destination port, and type of service bits. The flow record keeps information such as the number of

Issue	Sampled NetFlow	Adaptive NetFlow	Flow Slices
Memory usage	Variable	Fixed	Fixed
Volume of flow data reported	Variable	Fixed	Fixed
Behavior under DDoS with spoofed sources and other traffic mixes with many flows	Panicky flow expiration	Reduction in accuracy	Small reduction in accuracy
Estimates of traffic in small time bins	Less accurate	Accurate	Less accurate
Reporting overhead when using small bins	Unaffected	Large increase	Unaffected
Lifetime of flow record in router memory	Min (active timeout, flow length + inactivity timeout)	Bin length	Min (slice length, flow length + inactivity timeout)
Resource usage at end of time bin	N/A	Reporting spike or extra memory	N/A
Processing intensive tasks	Counting	Counting and renormalization	Counting
Counting TCP flow arrivals (using SYNs)	Yes	Yes	Yes
Counting all active flows	No	Separate flow counting extension	Yes
Counting all active flows at high speeds	No	Hardware flow counting extension	No

Table 1: Sampled NetFlow, Adaptive NetFlow and Flow Slices differ in the types of measurements they support, in how they adapt to different traffic mixes, and in their resource consumption (memory usage and reporting traffic).

packets in the flow, the (total) number of bytes in those packets, the timestamp of the first and last packet, and protocol flag information such as whether any of those packets had the SYN flag set. NetFlow uses four rules to decide when to remove a flow record from router memory and report it to the collection station: 1) when TCP flags (FIN or RST) indicate flow termination, 2) 15 seconds (configurable "inactive timeout") after seeing the last packet with a matching flow ID, 3) 30 minutes (configurable "active timeout")' after the record was created to avoid staleness and 4) when the memory is full.

On every new packet, NetFlow looks up the corresponding entry (creating a new entry if necessary) and updates that entry's counters and timestamps. Since for high speed interfaces, the processor and the memory holding the flow records cannot keep up with the packet rate, Cisco introduced Sampled NetFlow [22] which updates the flow cache only for sampled packets. For a configurable value of a parameter N, a packet is sampled with one in N probability.

One problem with NetFlow is that the memory required by the flow records and the bandwidth consumed to report them depends strongly on the traffic mix. In particular, large floods of small packets with randomly spoofed source addresses can increase memory and bandwidth requirements by orders of magnitude. Adaptive NetFlow [10] solves this problem by dynamically adapting the sampling rate. Adaptive NetFlow divides the operation of the flow measurement algorithm into equally spaced time bins. Within each bin, the algorithm starts by sampling aggressively (high sampling probability). If memory is consumed too quickly, it switches to less aggressive sampling. It then "renormalizes" existing entries so that they reflect the counts they would have had with the new sampling rate in effect from the beginning of the bin. At the end of the bin, all entries are reported.

Using fixed size bins in Adaptive NetFlow increases the memory utilization compared to Sampled NetFlow and causes bursts in reporting bandwidth. Memory utilization is higher because, to operate seamlessly between bin-boundaries, Adaptive NetFlow requires two sets of records (double-buffering), one for current bin and one for records in the previous bin while they are being transmitted. Without double-buffering, flow records that expire at the bin-boundary need to be transmitted immediately to create space for the next set of entries. Large flows spanning multiple bins are reported separately for every bin increasing the bandwidth usage. Table 1 gives a summary comparison of Sampled NetFlow, Adaptive NetFlow and Flow Slices.

The flow records are used to estimate the number of bytes or packets in various traffic aggregates of interest. This can give network operators information about dominant applications, the network usage of various clients, traffic matrices, and many other useful statistics [12, 19, 1, 14]. Smart Sampling [8] is a way of reducing the data used by such analyses without significantly affecting their results. Smart Sampling retains flow records with probability proportional to the size of their byte counter. The flow records can also be used to estimate the number of active flows which is important when looking for denial of service attacks, scans, and worms in the traffic mix. Unfortunately, if we use Sampled NetFlow it is impossible to recover the number of flows in the original traffic from the collected data [5] unless we use protocol information. By using the SYN flag information in flow records we can accurately estimate the number of TCP flows in the traffic mix [9].

3 Description of flow slices

The core flow slicing algorithm is based on the sample and hold algorithm [11]. After presenting the core algorithm, we discuss four extensions: adding packet sampling to scale to high speed links, using an inactivity timeout to reduce memory usage at a router, adding binned measurement to reduce binning error during analysis, and adding multi-factor smart sampling to control the volume of flow data reported. The version of Flow Slices described used for Table 1 has the first two extensions. We also discuss the configuration parameters of Flow Slices, and how they can be set adaptively based on the current traffic mix.

3.1 Core algorithm

The core flow slicing algorithm addresses the problem of reducing the memory usage of the flow measurement module. Sampled NetFlow and Adaptive NetFlow use random packet sampling: they only handle sampled packets. Just as sample and hold [11], flow slicing uses sampling only to control the creation of flow entries, once a sampled packet creates an entry for a flow, all its subsequent packets are counted (not just the sampled ones). This increases the accuracy of the estimates of packet counts, without changing the memory requirement. We use the "flow slicing probability" p to control the creation of flow entries. We expire and report each entry exactly t seconds after its creation, irrespective of the rate at which packets arrive for a particular flow. We call this core algorithm "flow slicing" because each entry tracks a "slice" of length t from the flow. Just as in the case of NetFlow, the entry associated with a flow has a byte and packet counter updated at every packet, timestamps for the first and last packet, and it stores protocol information such as whether any of the packets counted against the entry had the SYN flag set. To ensure unbiasedness of estimators, on creation of an entry we do not initialize the byte counter to the number of bytes b_{first} in the packet that caused the creation of the entry, but to b_{first}/p (see Section 4.2 for more details).

The slice length t is related to the "active timeout" of NetFlow which controls for how long an active entry is kept before expiring and being reported (default 30 minutes). Both of these parameters limit the staleness of the data (i.e. if we have a long-lived flow, we know that its traffic will be reported with at most this much delay).

By dynamically adapting the flow slicing probability, we can control the rate at which entries are created and freed, thus ensuring that the algorithm stays within its allocated memory budget M. By keeping the rate at which entries are created, on average slightly below M/t, we can also keep the rate at which flows records are reported smooth. In contrast Adaptive NetFlow proposes expiring all active entries at the end of the measurement bin, so it either has a large peak in reports, or it requires buffers that increase the memory usage by almost a factor of two if the reporting of the records is smoothed out over the next measurement bin. We do not however, discuss dynamic adaptation in much detail in this paper, as adaptation techniques similar to that in [10] can be applied in this context using feedback from the current memory usage. Note however, that in our adaptation, we do not require the costly operation of renormalization that is required in Adaptive NetFlow. Next we discuss some of the tuning knobs we provide to control the three resource bottlenecks (CPU, Memory, Bandwidth).

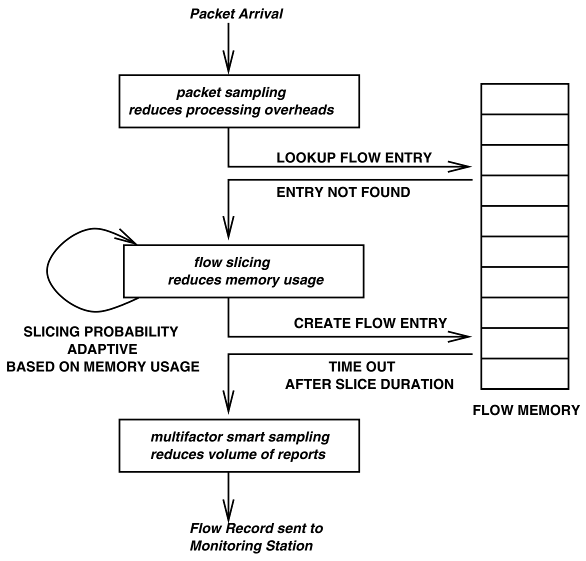

Packet Arrival
packet sampling reduces processing overheads
LOOKUP FLOW ENTRY
ENTRY NOT FOUND
flow slicing reduces memory usage
SLICING PROBABILITY ADAPTIVE BASED ON MEMORY USAGE
CREATE FLOW ENTRY
TIME OUT AFTER SLICE DURATION
FLOW MEMORY
multifactor smart sampling reduces volume of reports
Flow Record sent to Monitoring Station

Figure 1: Architecture

3.2 Scaling to high speeds

The flow slicing probability p controls the memory usage, but since we do a lookup in the flow memory for every packet, flow slicing does not control the processing load. In the presence of limited processing power, we add a random packet sampling stage in front of the flow slicing stage (see Figure 1). A simple solution is to set the packet sampling probability q statically to a value that ensures that the processor performing the flow measurement can keep up even with worst case traffic mixes. Based on Cisco recommendations [17] for turning on NetFlow sampling for speeds higher than OC-3, we set q to $1/4$ for OC-12 links, $1/16$ for OC-48, etc. With these packet sampling rates, and with worst case traffic consisting of the link entirely full with 40-byte packets, the flow measurement module has around $2\mu s$ per packet and it has time to perform around 35 (wide) DRAM accesses on average.

3.3 Adding an inactivity timer

Most flows in the Internet are short-lived. If our only mechanism for removing an entry is its expiration after the slice length t and we use a large value for t, at any moment in time, most of the entries in the flow memory will belong to flows that are no longer active and just use up memory waiting to expire. On the other hand having a very short slice length can lead to an increase in reporting traffic and loss of accuracy. Adding an inactivity timeout parameter $t_{inactive}$ to flow slices reduces the memory spent on obsolete entries. Experimental results in Section 6.1 show that we can significantly reduce the memory requirement by using inactivity timers. An adaptive algorithm for setting the flow slicing rate can turn this reduction in memory usage into an increase in accuracy.

3.4 Adding binned measurement

With flow slices we have the same problem as with NetFlow if we want to perform traffic analysis using time bins: for flow slices that span time bins, we can only guess how many of the flow's packets were in each bin, and this introduces errors in the results. This problem is even more pronounced when analysis is required in very small time bins to capture more precise traffic dynamics. We can extend flow slices to support binned measurement of traffic by keeping multiple sets of byte and packet counters, one set for each bin the slice passes through. By keeping separate counters for each bin, the binning error is eliminated entirely, at the cost of increasing the size of the flow records. Note that the reporting bandwidth costs of this solution are significantly smaller than those of the solution used by Adaptive NetFlow where an entire record is reported for each bin. The byte and packet counters are 8 bytes whereas a complete record is 48 bytes.

The number of counters per record has to be one larger than the number of bins required to fit a slice because the flow slice can overlap only partially with the first and last bin. The choice of the size of the measurement bin supported is a compromise between resource consumption at the router and accuracy of results. Reasonable choices can range anywhere from the slice length t to 20 times smaller. For brevity, we do not explore this further in the paper, but note that depending on the final goal, the flow slicing algorithm can be extended with additional resources to obtain the desired accuracy.

3.5 Controlling the reporting bandwidth

Smart sampling has been proposed as a way of reducing the number of flow records without causing much error. Smart sampling focuses on measuring the number of bytes in arbitrary aggregates of traffic and thus smart sampling favors flow records with large byte counters over those with small flow counters. Common packet sizes vary between 40 and 1500, so while the packet counts are not proportional to the byte counts, they are closely correlated. Thus smart sampling will ensure that the errors introduced in packet counts are also small. The situation is different with flow arrival counts. These depend heavily on flow records with the SYN flag set, and most such records come from small flows which are discriminated against by smart sampling. Thus the errors introduced by smart sampling in the flow arrival counts are significant.

We propose a new variant of smart sampling, *multi-factor smart sampling* which takes into consideration not just byte counts, but also packet counts and SYN flags. While multi-factor smart sampling still favors flow records with large byte and packet counts, it also favors records with the SYN flag, thus ensuring that the errors introduced into the flow arrival counts are not large either. Because the exact rule used to determine the multi-factor smart sampling probability r depends on estimators of byte and packet counts, we postpone its discussion to Section 4.5.

3.6 Setting the parameters of flow slicing

Routers or other network devices performing flow measurement have three types of resources that can become bottlenecks: processing power, flow memory, and reporting bandwidth. Flow slices use three different "tuning knobs" to control these three resources: the packet sampling probability q controls the processing load, the flow slicing probability p controls the memory usage and the thresholds determining the smart sampling probability r control the volume of data reported. This can result in more accurate traffic analysis results than using a single parameter, the packet sampling probability, to control all three resources, as Adaptive NetFlow does. This distinction would be irrelevant in practice if the only scarce resource would be the processing power at the router, so it is useful to perform a quick sanity check before proceeding any further: can an unfavorable traffic mix push the memory requirements or reporting bandwidth so high that they become a problem? First, let us assume a traffic mix consisting of back-to-back minimum sized packets, each belonging to a different flow (a massive flooding attack with randomly spoofed source addresses). With the packet sampling rates from Section 3.2, the traffic measurement module would receive a packet every $2\mu s$. Even with an aggressive inactivity timeout of $t_{inactive} = 5$ seconds, we need a flow memory that can fit $2,500,000$ flow records, which at 64 bytes/record[17] requires 153 megabytes. When reported flow records take 48 bytes (ignoring overheads), so at $500,000$ flow records/second, which requires 192 megabits/second. These numbers are orders of magnitude above what one can comfortably afford. The experiments from Section 6 use realistic traffic mixes to evaluate the benefits of Flow Slices as compared to Sampled NetFlow and Adaptive NetFlow.

For each of the parameters of Flow Slices listed in

Parameter	What it controls	How it is set
Flow slicing probability	Memory usage at router	Adaptively based on memory usage
Flow slice length	Staleness of reported data	Statically based on user preferences
Inactivity timeout	Reduces memory usage	Statically based on typical inter packet arrival time
Packet sampling probability	Processing load at router	Statically based on worst case traffic
Bin size (optional)	Binning error	Statically based on user preferences
Smart sampling thresholds	Volume of flow data reported	Adaptively or statically based on target volume

Table 2: Configuration parameters for Flow Slices.

Table 2, we need to decide whether to set them statically as part of the router configuration, or dynamically adapt them to the current traffic mix. Of the three main tuning knobs, the flow slicing probability p should definitely be set dynamically to allow the router to protect from memory overflow when faced with unfavorable traffic mixes. The thresholds controlling the smart sampling probability can also be set adaptively. In this paper, we consider that the packet sampling probability q is static based on recommended values for different link capacities. Flow Slices would work just as well with a dynamic packet sampling probability that could go above the conservative static value, but since it is hard to guarantee the stability of such an approach without pushing the packet sampling rate adaptation logic into hardware (which raises deployment problems), we chose not to explore such a solution here.

The observant reader might have noticed that without the optional binned measurement feature Flow Slices resembles Sampled NetFlow. If the dynamic adaptation algorithms set the flow slicing probability p and the smart sampling probability r to 1 the two solutions perform exactly the same processing. We consider this to be an important feature. The difference between Sampled NetFlow and Flow Slices is in how they react to unfriendly traffic mixes and environments with strong constraints on resources. While both Adaptive NetFlow and Flow Slices provide robustness to unfavorable traffic mixes, Adaptive NetFlow forces the user to adopt the binned measurement model (which can increase memory usage and the volume of reports) even when the traffic mix is favorable.

4 Estimators based on flow slices

In this section, we discuss formulae for estimating traffic based on the flow records provided by Flow Slices. In practice, the user would be interested in the number of bytes, packets or flows in the entire traffic mix or a portion of it (e.g. the HTTP traffic, etc.). All our estimators focus on a single flow. To compute the total traffic, the user has to sum the contributions of all individual flow records. If the estimators for individual flows have the property of unbiasedness, the errors in the estimates for individual flows will not accumulate, but cancel out (to some extent).

For the purposes of our analysis, a bin is an arbitrary interval of time of interest to traffic analysis. To simplify analysis, we start by focusing on the simple case of a single bin, with slice length t and inactivity timeout $t_{inactive}$ larger than the size of the bin and flow memory empty at the beginning of the bin. Next, we look at how the estimators generalize when we remove these constraints. Table 3 summarizes notation used throughout the paper.

4.1 Estimating packet counts

The packet counter c_s in an entry is initialized to 1 when the first packet of the flow gets sampled, and it is incremented for all subsequent packets belonging to the flow. Let s be the number of packets in the flow at the input of the flow slicing algorithm. Equation 1 gives the formula for our estimator $\widehat{s}$ for the number of packets in the flow.

$$\widehat{s} = 1/p - 1 + c_s \tag{1}$$

Lemma 1 *$\widehat{s}$ as defined in Equation 1 is an unbiased estimator of s.*

Proof: By induction on the number of packets s.

Base case: If $s = 1$, the only packet of the flow is sampled with probability p and in that case it is counted as $1/p - 1 + 1 = 1/p$ packets. With probability $1 - p$ it is not sampled (and it counts as 0). Thus $E[\widehat{s}] = p \cdot 1/p + 0 = 1 = s$.

Inductive step: By induction hypothesis, we know that for a flow with $s' = s - 1$, $E[\widehat{s'}] = s' = s - 1$. Also since the flow slice length t and the inactivity timeout $t_{inactive}$ are larger than the bin size, we know that once the flow gets an entry, all its packets within the bin will get counted by c_s. There are two possible cases: the first packet of the flow gets sampled, and we get $c_s = s$, or it doesn't and then the value of c_s and $\widehat{s}$ will be the same as those for a flow with $s' = s - 1$ packets for which the sampling decisions are the same as for the rest of the packets of our flow.

$$\begin{aligned} E[\widehat{s}] &= p \cdot (1/p - 1 + s) + (1 - p)E[\widehat{s'}] \\ &= 1 - p + ps + (1 - p)(s - 1) = s \end{aligned}$$

■

Name	Meaning
p	flow slicing probability
q	packet sampling probability
r	smart sampling probability
s	size of flow (in packets) before flow slicing
c_s	packet counter in flow record
$\widehat{s}$	estimate of the size of flow before flow slicing (0 if flow not sliced)
S	original size of flow (in packets) before packet sampling
$\widehat{S}$	estimate of the original size of flow (0 if flow not sampled or not sliced)
b	size of a flow in bytes before flow slicing
c_b	byte counter in flow record
$\widehat{b}$	estimate of the number of bytes in flow based on flow slices (0 if flow not sliced)
B	original size of flow in bytes before packet sampling
$\widehat{B}$	estimate of the original size of flow in bytes (0 if flow not sampled or not sliced)
$\widehat{f}$	contribution to the estimate of the number of active flows (0 if flow not sliced)
$\widehat{a}$	contribution to the estimate of the number of flow arrivals (0 if flow not sliced)
$\widehat{A}^{(1)}$	contribution to first estimator of number of flow arrivals (0 if flow not sampled or not sliced)
$\widehat{A}^{(2)}$	contribution to second estimator of number of flow arrivals (0 if flow not sampled or not sliced)
z_s	smart sampling threshold controlling the influence of $\widehat{S}$ on r
z_b	smart sampling threshold controlling the influence of $\widehat{B}$ on r
z_a	smart sampling threshold controlling the influence of $\widehat{A}^{(1)}$ on r

Table 3: Notation used in this paper.

If we sample packets randomly with probability q before applying the flow slicing algorithm, we will want to estimate the number of packets S at the input of the packet sampling stage. Since $E[s] = qS$, it is easy to show that $\widehat{S} = 1/q\widehat{s}$ is an unbiased estimator for S.

4.2 Estimating byte counts

Before discussing how to estimate byte count estimates in flow slices, we show why a simpler solution does not work. We could have the byte counter c_b in the flow entry just count the total number of bytes in the packets seen once the flow record is created. Just like with the packet counter, we need an additive correction to account for the packets missed before the creation of the entry. We can get an unbiased estimate for the number of packets missed, but not for their total size, because we do not know their sizes. We could assume that the packet sizes are uniform within the flow, but this would lead to systematic biases because they are not. As the proof of Lemma 2 shows, storing the size of the sampled packet that led to the creation of the entry would solve the problem because using it to estimate the total number of bytes in the packets not counted does lead to an unbiased estimator. But this would require another entry in the flow record. Instead, we store this information in the byte counter itself by **initializing** c_b **to** b_{first}/p when the entry is created (b_{first} is the size in bytes of the sampled packet). Let b be the number of bytes of the flow at the input of the flow slicing algorithm.

$$\widehat{b} = c_b \tag{2}$$

Lemma 2 *$\widehat{b}$ as defined in Equation 2 is an unbiased estimator of b.*

Proof: By induction on the number of packets in the flow s. Let b_i for i from 1 to s be the sizes of the individual packets. By definition the number of bytes in the flow is $b = \sum_{i=1}^{s} b_i$. For convenience of notation, we index the packet sizes in reverse order, so b_1 will be the size of the last packet and b_s the size of the first one.

Base case If s=1, the only packet is sampled with probability p and in that case it is counted $c_b = b_1/p = b/p$ bytes. With probability $1 - p$, it is not sampled (and it counts as 0). Thus $E[c_b] = p \cdot b/p + 0 = b$.

Inductive step By induction hypothesis, we know that if the first packet is not sampled we are left with the last $s' = s - 1$ packets and $E[c_b] = b' = b - b_s$. If the first packet gets sampled, we count it as b_s/p and we count the rest exactly because the flow slice length t and the inactivity timeout $t_{inactive}$ are larger than the bin size.

$$\begin{aligned} E[c_b] &= p \cdot (b_s/p + b') + (1-p)b' \\ &= b_s + pb' + (1-p)b' = b_s + b' = b \end{aligned}$$

■

If we sample packets randomly with probability q before applying the flow slicing algorithm, we will want to estimate the number of bytes B at the input of the packet sampling stage. Since $E[b] = qB$, it is easy to show that $\widehat{B} = 1/q\widehat{b}$ is an unbiased estimator for B.

4.3 Estimating the number of active flows

We use two definitions for counting flows: active flows and flow arrivals. A flow is active during a time bin if it sends at least one packet during that time bin. Consecutive TCP connections between the same two computers that happen to share the same port numbers are considered a single flow and they will be reported in the same flow record under our current assumptions. Active flows with none of their packets sampled by the flow slicing process, will have no records; at least some of the flow records we get should be counted as more than one active flow, so that the total estimate will be unbiased. We count records with a packet counter c_s of 1 as $1/p$ flows and other records as 1 flow and this gives us unbiased estimates for the number of active flows.

$$\widehat{f} = \begin{cases} 1/p & \text{if } c_s = 1 \\ 1 & \text{if } c_s > 1 \end{cases} \tag{3}$$

Lemma 3 *$\widehat{f}$ as defined in Equation 4 has expectation* 1.

Proof: There are three possible cases: if a packet before the last gets sampled, $c_s > 1$, if only the last packet gets sampled $c_s = 1$, and if none of the packets gets sampled there will be no flow record, so the contribution of the flow to the estimate of the number of active flows will be $\widehat{f} = 0$. The probability of the first case is $p_{s-1} = 1 - (1-p)^{s-1}$, the probability of the second is $p(1 - p_{s-1})$ and that of the third is $(1-p)(1-p_{s-1})$.

$$\begin{aligned} E[\widehat{f}] &= p_{s-1} \cdot 1 + p(1-p_{s-1}) \cdot 1/p + \\ &\quad (1-p)(1-p_{s-1}) \cdot 0 = 1 \end{aligned}$$

■

The estimators for the number of bytes and packets in a flow were trivial to generalize to the case where we apply random packet sampling before flow slicing because the expected number of packets and bytes after packet sampling was exactly q times the number before. For the number of active flows there is no such simple relationship and actually it has been shown that it is impossible to estimate without significant bias the number of active flows once random sampling has been applied [5]. But by changing slightly the definition of flow counts we can take advantage of the SYN flags used by TCP flows.

4.4 Estimating flow arrivals

Flow arrivals are defined only for TCP flows which should start with one SYN packet. A flow is considered to have arrived in a bin if its SYN packet is in that time bin. Flows active during a certain bin, but with their SYN packet before the bin do not count as flow arrivals for that bin (but they count as active flows). If we look a the core flow slicing algorithm we can use the following estimator to compute the number of flow arrivals.

$$\widehat{a} = \begin{cases} 1/p & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases} \tag{4}$$

Given that the SYN flag is set in the flow record if it was set in *any* of the packets counted against the record, it is trivial to prove that $\widehat{a}$ leads to unbiased estimates of the number of flow arrivals if we make an assumption.

Assumption 1 *Only the first packet for the flow can have the SYN flag set.*

The flow arrival information is preserved by random packet sampling. Duffield et al. propose two estimators of the number of flow arrivals that work based on flow records collected after random sampling of the traffic [9]. The formulas for the individual contributions of flow records to the total estimate of the number of flow arrivals are as follows.

$$\begin{aligned} \widehat{M}^{(1)} &= \begin{cases} 1/q & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases} \\ \widehat{M}^{(2)} &= \begin{cases} 1/q & \text{if SYN flag set and } s = 1 \\ 1 & \text{if SYN flag not set or } s > 1 \end{cases} \end{aligned}$$

Duffield et al. show [9] that both estimators are unbiased $E[\widehat{M}^{(1)}] = E[\widehat{M}^{(2)}] = 1$ for flows that have exactly one SYN packet. Both estimators overestimate the number of flow arrivals if flows have more than 1 SYN packet. For flows without any SYN packets which according to our definition of flow arrivals (which differs slightly from that used in [9]) should not be counted, we have $E[\widehat{M}^{(1)}] = 0$ and $E[\widehat{M}^{(2)}] > 0$, so to make the second estimator unbiased we need another assumption.

Assumption 2 *The first packet within the bin for every flow has the SYN flag set.*

Flows retaining SYN packets after the random packet sampling stage will retain a single SYN packet, and $\widehat{M}^{(1)}$ estimates the number of flow arrivals based on the number of such flows. We can easily combine it with $\widehat{a}$ to get an estimator for the number of flow arrivals for the combined algorithm using random packet sampling and flow slicing.

$$\widehat{A}^{(1)} = \begin{cases} 1/(pq) & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases} \tag{5}$$

$\widehat{M}^{(2)}$ treats separately flows that only have a SYN packet after packet sampling and the others that survive it. Fortunately we can differentiate between the two types of flows even after flow slicing is applied: if a flow with a single

SYN packet is sampled by flow slicing its record will have $c_s = 1$ and the SYN flag set; if any other flow is sampled by flow slicing and it has $c_s = 1$ at the end of the bin it means that only its last packet was sampled thus it will not have the SYN flag set because that would put it into the category of flows with a single SYN packet surviving the packet sampling. Thus we can combine $\widehat{M}^{(2)}$ with $\widehat{a}$ to obtain another estimator.

$$\widehat{A}^{(2)} = \begin{cases} 1/(pq) & \text{if SYN flag set and } c_s = 1 \\ 1/p & \text{if SYN flag not set and } c_s = 1 \\ 1 & \text{if SYN flag not set and } c_s > 1 \end{cases} \quad (6)$$

Note that if assumption 1 is violated and we have more than one SYN packet at the beginning of the flow, say due to SYN retransmissions, both estimators will be biased towards over-counting. But if repeated SYNs are a rare enough occurrence, the effect on a final estimate based on many flow records will be small.

4.5 Multi-factor smart sampling

To reduce the number of flow records, while maintaining accurate byte counts, smart sampling [8] proposes sampling the flow records with a size dependent probability $r = \min(1, b/z)$ where z is a threshold parameter controlling the trade-off between the loss in accuracy and the reduction in the volume of reports. We can adapt smart sampling to flow slices using $r = \min(1, \widehat{B}/z)$ and we could still estimate byte, packet and flow arrival counts based on the smart sampled flow records using $\widehat{\mathcal{S}} = 1/r\widehat{S}$, $\widehat{\mathcal{B}} = 1/r\widehat{B}$, and $\widehat{\mathcal{A}} = 1/r\widehat{A}$. But using this formula for r results in a variance for $\widehat{\mathcal{A}}$ much larger than that of $\widehat{A}$ because it discriminates against flows with few bytes, and since most flows have few bytes, they will also produce most flow records with the SYN flag set – and these are exactly the records $\widehat{A}^{(1)}$ and $\widehat{A}^{(2)}$ rely on.

We propose a new variant of smart sampling, multi-factor smart sampling, which takes into consideration not just byte counts, but also packet counts and SYN flags. By picking a smart sampling probability of $r = \min(1, \widehat{s}/z_s + \widehat{B}/z_b + \widehat{A}/z_a)$ we can balance the requirements of the three estimators. The three individual thresholds control the trade-off between accuracy and reduction in report volume separately for the three estimators of bytes, packets and flow arrivals. Note that multi-factor smart sampling is a generalization of smart sampling: if we set $z_b = z$, $z_s = \infty$, and $z_a = \infty$, it will assign the exact same sampling probabilities to records as smart sampling.

4.6 Dynamically adjusting the flow slicing probability

Flow Slices dynamically adjusts the flow slicing probability p to the current traffic. This adjustment can happen in the middle of a time bin. Which one of the many values of p should we use in our estimators? Are the estimators still unbiased? Actually none of the proofs depends on having a single value for p, and they would all work if we replaced it with a separate p_i for every packet. All the estimators would need to use the value of the packet slicing probability in effect at the time the sampling of a packet caused the creation of the entry. This doesn't necessarily mean that one needs to extend the flow entry with one more field, because it already holds the timestamp of the first packet and that can be used to determine the flow slicing rate if the router keeps a small log of recent adjustments to it.

When the flow record expires and it is reported, the report should include the value of the flow slicing probability p in effect at the time the entry was created. Similarly if the smart sampling thresholds z_s, z_b, and z_a are adjusted dynamically, the report should include their current value so that one can compute r during analysis. If one uses just a few possible values for these parameters (e.g. only powers of two), each of these sampling rates can be encoded in less than one byte, so the reporting overhead they impose is limited (a flow record has 48 bytes).

4.7 Bins, timeouts, and flow reconstruction

To simplify our discussion of the estimators we started with some strong assumptions: all records last longer than the bin length, counters count only packets within the bin of interest, and the flow memory is empty at the beginning of the bin. In this section we relax these assumptions and discuss the effects of these relaxations on the estimators.

4.7.1 Continuous operation

The most elementary relaxation of the assumption is to consider continuous operation of the algorithm: records still last longer than the bin length, and we still have separate counters for each bin, but there can be active records at the start of our bin, records created earlier.

The simplest case is that of records spanning the entire bin. The byte and packet counters will reflect the actual traffic, so we use $\widehat{S} = 1/qc_s$ and $\widehat{B} = 1/qc_b$. If we do not have a packet sampling stage we can also compute $\widehat{f} = 1$ if $c_s > 0$ and $\widehat{f} = 0$ otherwise. $\widehat{A} = 0$ because the flow started in an earlier bin.

If a flow record expires within the bin we run the analysis on, it can be the only record for the flow, but it is also possible that another record for the same flow would get created after the first record's expiration. For byte and packet counts which are additive we can just add the counters from the first record to the estimates from the second $\widehat{s} = \widehat{s}_1 + \widehat{s}_2$ and $\widehat{b} = \widehat{b}_1 + \widehat{b}_2$. The analysis of unbiasedness carries through because we can consider that the bin is actually two sub-bins, one ending when the first record ends and the other starting at the same time. Since we have unbiased byte and packet estimates for both sub-bins, our estimates for the sum of the bins will still be unbiased.

If $c_{s1} > 0$, we know that the flow sent packets during the bin, so we set $\widehat{f}$ to 1, otherwise we use Equation 3 with c_{s2} since an unbiased estimator for whether the flow was active in the second sub-bin will tell use whether it was active overall. This approach preserves overall unbiasedness, but it makes analysis more complicated because the two flow records representing the flow cannot be processed independently anymore: the contribution of the second record to the flow count of the bin depends on whether there was a first record with the same flow identifier. When the router reports the records, they might not be near each other, so the analysis has to do "flow reconstruction": keep a hash table with flow identifiers and find flow records with the same flow identifier covering parts of the same bin. The consequence of not doing flow reconstruction is running the risk of double counting such flows with more than one record (which might be acceptable in many settings).

By our definition of flow arrivals from Section 4.4, as long as assumption 1 holds, if a flow has a record that starts before the start of the bin, we should use $\widehat{A} = 0$, irrespective of whether we have a second flow record (possibly with a SYN flag) or not. If we have a second flow record with the SYN flag set we can clearly say that assumption 1 does not hold, but without flow reconstruction we might count it separately against the flow arrival count. In many settings this type of over-counting is not a serious concern. $\widehat{A}^{(2)}$ should not be used because assumption 2 does not hold.

4.7.2 Slices shorter than bins

When the inactivity timeout $t_{inactive}$ is short or when the analysis is over long time bins (say hours), flow slices can be shorter than the bin size. It can happen that we have more than two records for the same flow within the same bin. For byte and packet counts we can just add the individual estimates for the different records and we get an unbiased estimator for the entire bin. For active flows we cannot get an unbiased estimate, not even with flow reconstruction. For flow arrivals, by using $\widehat{A}^{(1)}$ for the individual records and summing the contributions without any flow reconstruction gives unbiased estimates as long as assumption 1 is not violated. For a record started before the beginning of the bin, even if it has the SYN flag set in violation of assumption 1 we do not count it as flow arrival and thus have $\widehat{A}^{(1)} = 0$.

4.7.3 Binning errors

So far we assumed that Flow Slices uses binned measurement. This guarantees that as long as the analysis is on time intervals that are exact multiples of the measurement bins used, it will be easy to determine exactly how many of the packets and the bytes counted by the record were within the bin. But by default Flow Slices doesn't use bins, and for records that span bin boundaries, the user will have to guess how the packets and bytes were actually divided between the bins. We can prove that our reconstruction of how the traffic divides between the bins is unbiased only if we make an assumption about the spacing of the packets.

Assumption 3 *For every flow at the input of the flow slicing algorithm, the time between the arrivals of all pairs of its consecutive packets is the same.*

We use the following algorithm for distributing the packets of reported by a flow record that spans bins between the bins covered by the record. We consider c_s packet arrival events, the first one is the timestamp of the first packet counted by the entry, the last one the timestamp of the last packet counted by the entry and the remaining $c_s - 2$ evenly spaced between them. We consider that 1 packet arrived at every packet arrival event, except for the first event which has $1/p$ packets, and distribute the packets between bins accordingly. This can be shown to be an unbiased way of distributing packets between bins under assumption 3. We recommend distributing the c_b bytes of the flow between bins proportionally with the number of packets counted against each bin. Assumption 3 is not enough to prove this distribution of bytes between the bins to be unbiased, we would need an additional assumption about uniformity of packet sizes. For flow arrivals, we do not have a binning problem because we assume that the first packet counted by the flow record is the one with the SYN, so we count the flow arrival against the bin the first packet is in.

We cannot achieve provably unbiased binning for bytes and packets under realistic assumptions about inter packet arrival times and packet size distributions within flows. We turn to measurements instead to see how much the binning error is on typical traffic. We recommend using such experimental results to decide whether increasing the size of the flow record by adding multiple counters to do binned measurement is worth it.

5 Variances of estimators

The estimators discussed in the previous section were all defined on an individual flow and to compute a measure (say the number of packets) for a larger aggregate, the analyst would sum the values of the estimators for the flow records matching the aggregate. The sampling decisions for different flows are fortunately independent and thus the variance of the estimates for aggregates are the sum of the respective variances for the estimators for individual flows. In this section we focus on studying the variances of the various estimators for individual flows. We also show that the variances of the estimators based on the core flow slicing algorithm are lower than those of estimators based on random sampling used by Adaptive NetFlow to control memory usage. As in Section 4, we start with a simplified setting of a single bin in isolation and then proceed to more realistic settings. The proofs for the variance results from this section can be found in technical report[15].

5.1 Packet count variance

For the core flow slicing algorithm we can compute the variance of the packet count estimator.

$$VAR[\widehat{s}] = 1/p(1/p-1)(1-(1-p)^s) \qquad (7)$$

Note how this variance is strictly lower than the variance of results based on random packet sampling $(1/p-1)s$ except for the case of $s=1$ when the two variances are equal. The higher s, the larger the difference between the variance of results based on flow slicing when compared with packet sampling. Since using the same sampling probability will give the same memory usage for flow slicing and ordinary sampling, this comparison of variances shows us that flow slicing is a superior solution. The advantage is most apparent when estimating the traffic of aggregates with much traffic coming from large flows.

The same conclusion holds if we compare the combination of packet sampling and flow slicing used by Flow Slices to the pure packet sampling used by Adaptive NetFlow and Sampled NetFlow. Here the fair comparison is with Sampled NetFlow using a packet sampling probability of pq. We can conceptually divide this into a first stage of packet sampling that samples packets with probability q and a second one that samples them with probability p. The first stage has identical statistical properties for the two solutions, thus the difference in the accuracy is given by the second stage, but comparing the second stages reduces to comparing flow slicing and packet sampling using the same probability p.

5.2 Byte count variance

We can also compute the variance of the estimates for the number of bytes (we number the packet sizes b_i in reverse order with b_1 being the size of the last packet and b_s that of the first one).

$$VAR[\widehat{b}] = 1/p\sum_{i=1}^{s}(1-p)^{i-s+1}b_i^2 \qquad (8)$$

Note how this variance is strictly lower than the variance of results based on random packet sampling $(1/p-1)\sum_{i=1}^{s} b_i^2$ (except for the case of a single packet flow). This shows that for byte counts too, flow slices are a better solution than ordinary sampling.

5.3 Flow count variance

We can also compute the variance of the estimates for the number of active flows. We cannot compare against packet sampling because there are no unbiased estimates for the number of active flows based on packet sampled data.

$$VAR[\widehat{f}] = (1-p)^{s-1}(1/p-1) \qquad (9)$$

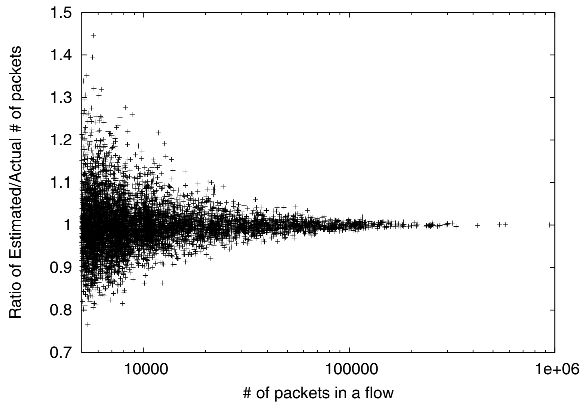


Figure 2: Scatter plot depicting the accuracy of packet count estimates based on flow slices.

5.4 Continuous operation

If we consider continuous operation for the algorithm, we can have at the beginning of the bin a record for our flow. If the slice spans the entire bin, it counts everything exactly and thus the variance of all estimator is 0. If the slice ends in the current bin, we can divide the flow into two parts: one covered by this older record and the rest. For the first part we have 0 variance for the byte and packet counts and for the second part we can apply formulas 7 and 8, but instead of s being the number of packets of the flow in the bin, it should be only the number of packets in this second part and the b_i be the sizes of those packets. For the flow count estimate, if the number of packets in the first record is 0, the variance of the estimate is 0, otherwise formula 9 applies. Thus having flow records active at the beginning of the bin does not increase the variance of the packet, byte and flow count estimates, on the contrary, it can reduce them significantly.

6 Experimental evaluation

We divide the experimental evaluation into two parts. The first group of experiments evaluates the efficacy of the core flow slicing algorithm and the multi-factor smart sampling algorithm. The second group compares the Flow Slices solution with Adaptive NetFlow (ANF) to show the efficacy of Flow Slices both in terms of memory usage and accuracy of estimates. For our evaluations, we obtained traces on an OC-48 link from CAIDA [4].

6.1 Accuracy of core flow slicing algorithm

In this section, we evaluate the core flow slicing algorithm against the "full-state" approach. These experiments provide more insight into the efficacy of the flow slicing algorithm and the effect of changing various variables such as flow slicing probability and slice length on the memory usage and the mean relative error of results.

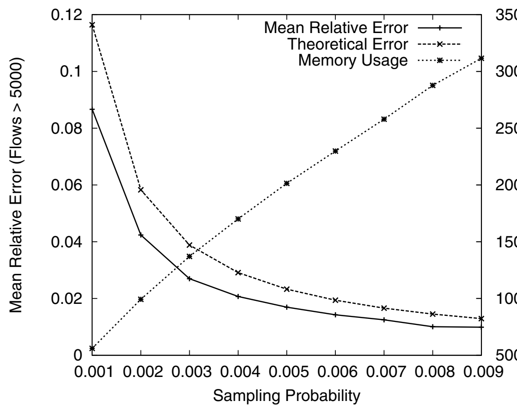


Figure 3: Trade-off between the mean relative error and memory usage as we increase flow slicing probability.

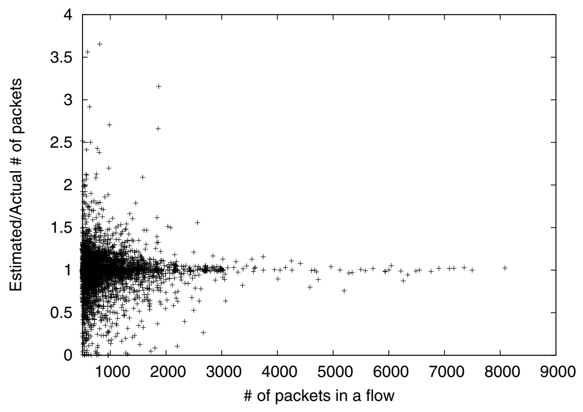


Figure 4: Scatter plot depicting the errors introduced in interpolating bin measures from slices.

Are estimates unbiased? For this experiment, we fix the flow slicing probability p to 0.8% (1 in 125) and the slice duration to 60 seconds. Figure 2 shows the scatter plot of ratio of the estimated and true flow sizes (in number of packets) on the y-axis with increasing true flow size on the x-axis. Note that the plot only shows flows that have more than 5,000 packets throughout the duration of the trace (1 hour). From this scatter plot, we can see that most of the flows have been accurately estimated (within 10%). The estimates converge to the true values as the flow size increases. The presence of two-sided errors empirically confirms the unbiasedness of estimates based on flow slicing.

What is the effect of flow slicing probability on the accuracy of these estimates ? According to Equation 7, increasing flow slicing probability increases the accuracy of estimated flow sizes. Besides, the memory usage should increase as the slicing probability increases. In Figure 3, the mean relative error for flows larger than 5,000 and the corresponding memory usage have been plotted with varying slicing probability on the x-axis. Apart from the empirical value of the mean relative error, we also plot the theoretical value based on Equation 7. Figure 3 confirms that increasing slicing probability decreases the mean relative error while increasing the memory usage. It can also be observed from the figure that the theoretical and empirical values of mean relative errors are in close agreement thus validating the analysis in Section 5.1.

What kinds of errors do we introduce by interpolating the number of packets in time bins? The goal of this experiment is to study the errors introduced when interpolating the number of packets in various time bins from flow slices that do not use bins (they only store the timestamps of the first and last packet). In Figure 4, the y-axis has the ratio of estimated to actual size of the flow in a given bin and the x-axis has the actual flow size (in packets). For this experiment, we used a slice length of 90 seconds and divided it up equally into 10 bins of size 9 seconds each. Two conclusions can be drawn from the results in Figure 4. First, for large flows, the error in the estimates obtained by interpolating bins from slices is insignificant. On the other hand, for relatively small flows, interpolating from flow slices results in much higher error. This is because we divide the entire volume of traffic for a particular flow among the bins it covers (see Section 4.7.3 for more details); the error depends on the timing of bursts of traffic. Of course, to capture the fine grained traffic information, the extension proposed in Section 3.4 could be used, but it would result in higher memory requirements. Second, we can observe the presence two-sided errors indicates lack of bias.

What is the effect of multi-factor smart sampling on the accuracy of estimates? In Section 4.5, we proposed a modification to smart sampling to improve the accuracy of the estimates for the number of flow arrivals. Table 4 summarizes the results of our experiment comparing multi-factor smart sampling with smart sampling. Before we discuss the details of this experiment, we want to note that we found that Assumption 1, that only the first packet of a flow can have the SYN flag set, is often violated in our trace. For some applications, the average number of packets with the SYN flag set per flow is almost 2 (due to SYN retransmissions). This affects all estimators of flow arrivals based on SYN counts. In this experiment, we do not aim to evaluate the accuracy of estimators based on SYN counts, but the effect of smart sampling on their estimates. Therefore, we do not measure the error relative to the actual number of flow arrivals, but to the estimate of flow arrivals based on the input to the two smart sampling algorithms. The input we used is the result of Flow Slices with a packet sampling probability of $q = 1/4$ and flow slicing probability of $p = 1/4$, using a slice length of 60 seconds and an inactivity timeout of 15 seconds. The threshold used for smart sampling is $z = 50,000$ bytes. The thresholds used for multi-factor smart sampling, $z_s = 1,000$ packets, $z_b = 500,000$ bytes and $z_a = 50$ flows, have been selected so that it produces

Port number used as aggregation key	Multifactor s. s. error			Smart sampling error			Actual traffic		
	Pkts	Bytes	SYNs	Pkts	Bytes	SYNs	Pkts	Bytes	SYNs
Web (80)	0.4%	0.7%	0.8%	0.3%	0.1%	1.6%	17.5M	1582M	1852K
Kazaa (1214)	0.4%	0.2%	2.4%	0.6%	0.1%	12.4%	2.67M	1527M	44.9K
eDonkey (4662)	0.5%	0.7%	1.5%	1.0%	0.2%	4.5%	2.96M	1075M	344K
telnet (23)	0.6%	0.8%	4.9%	0.9%	1.0%	39.2%	1.84M	79.1M	12.0K
SMB (445)	1.3%	1.6%	1.1%	2.5%	1.8%	3.1%	1.50M	93.3M	1380K
SMTP (25)	1.9%	1.0%	1.4%	2.7%	0.9%	6.4%	0.43M	130M	86.9K
DNS (53)	1.8%	2.4%	3.6%	2.7%	1.7%	16.8%	0.45M	34.8M	6.02K

Table 4: Comparison of the error introduced by multifactor smart sampling and smart sampling into estimates of traffic of various applications (average of 10 runs with different seeds). Both algorithms were configured to reduce the number of flow records from 1,700,000 to around 190,000.

Port number/ Range	Adaptive NetFlow		Flow Slices(60s)		Flow Slices(180s)		Flow Slices(300s)	
	Packets	Bytes	Packets	Bytes	Packets	Bytes	Packets	Bytes
Web (80)	0.5%	1.4%	0.4%	2.0%	0.5%	1.5%	0.3%	0.9%
Kazaa (1214)	1.2%	2.6%	1.0%	2.4%	0.8%	1.0%	1.0%	1.4%
eDonkey (4662)	1.4%	3.3%	1.7%	2.1%	1.1%	1.9%	1.0%	1.9%
telnet (23)	1.3%	1.5%	2.2%	2.2%	2.1%	1.8%	2.3%	2.5%
SMB (445)	2.6%	3.3%	2.5%	5.0%	2.2%	2.5%	1.7%	4.1%
SMTP (25)	1.9%	8.7%	2.3%	7.7%	3.9%	6.5%	3.8%	6.8%
DNS (53)	2.7%	4.0%	3.6%	2.6%	2.8%	3.6%	4.4%	4.9%
$>$ 50,000	9.7%	10.1%	5.4%	5.6%	3.7%	3.9%	3.1%	3.3%
10,000-50,000	20.5%	21.9%	15.3%	16.7%	12.2%	13.5%	10.8%	11.9%
5,000-10,000	30.6%	35.8%	26%	29.4%	22.2%	24.9%	19.9%	22.5%

Table 5: Comparison of the accuracy of estimates based on Adaptive Netflow and Flow Slices with different slice lengths.

approximately same number of records as smart sampling (from 1,700,000 down to roughly 190,000). Table 4 shows the error introduced by the two variants of smart sampling into estimates of the traffic of various applications identified by destination port numbers. While smart sampling introduces very large errors in the flow arrival estimates, multi-factor smart sampling ensures that the errors are comparable to packet and byte count estimates. For example, smart sampling incurs an error of 39.2% for telnet because it's small flows (approximately 6,600 bytes per flow on average compared to 34,000 for Kazaa) are discriminated against by smart sampling. Multi-factor smart sampling, on the other hand, achieves more accurate flow arrival counts by biasing its sampling towards records with non-zero flow arrivals. This typically results in only a slight reduction in the accuracy of packet and byte count estimates.

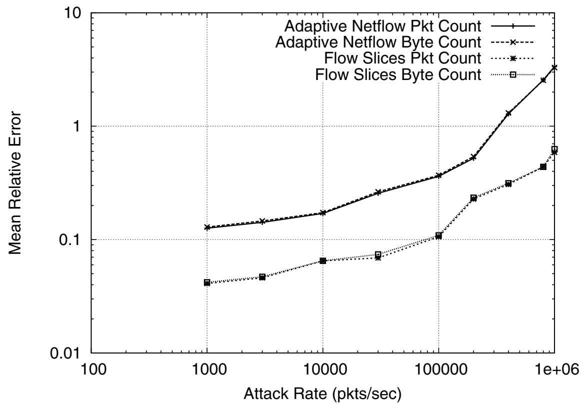
6.2 Comparison with Adaptive NetFlow

In this section, we compare Flow Slices with Adaptive NetFlow [10], a previously proposed solution based on packet sampling. For the purposes of evaluation, we fix the packet sampling probability to 1 in 1024 for ANF. To be fair in our comparisons with Flow Slices, we split the $1/1024$ probability into two parts consisting of packet sampling ($1/16$ for our OC-48 trace) and flow slicing probability ($1/64$). We compare average error in the estimates for both individual flows (categorized by ranges) as well as aggregates based on destination port number. typically, are Table 5 shows that ANF and Flow Slices have similar errors when estimating the traffic of various applications (aggregated by port). However, Flow Slices performs better than ANF (by about 10%) in the average error for individual flows. Varying the slice length from 60 to 300 seconds for Flow Slices did not affect the accuracy of the results significantly, although bigger slice lengths seem to perform a little better than with smaller slice lengths.

How does Flow Slices compare with ANF in resource consumption ? Table 6 summarizes the memory usage at the router and the volume of traffic reports for Flow Slices and ANF . Without an inactivity timeout, the resource requirements of the two solutions are similar. As we move to longer bins/slices there is a slight decrease in report volumes and a significant increase in memory requirements. Adding an inactivity timeout of 15 seconds to Flow Slices has a dramatic effect. The memory requirements are reduced significantly (about 80%) at the cost of only a slight increase in the volume of the reports (about 5%). With the

Trace	Packets	Slice length /	Memory (entries)			Report volume (records)		
	per second	Bin size (in secs)	Slices	$t_{inactive}$	ANF	Slices	$t_{inactive}$	ANF
1 (1 hour)	23,733	60	1,148	597	1,195	68,537	63,658	64,764
1 (1 hour)	23,733	180	3,021	741	3,141	61,316	57,028	60,229
1 (1 hour)	23,733	300	4,691	793	4,158	57,635	53,953	58,730
2 (10 mins)	124,988	60	5,378	3,065	5,641	25,896	26,362	27,509
2 (10 mins)	124,988	180	14,046	3,944	14,049	22,896	23,800	23,994
2 (10 mins)	124,988	300	21,667	4,218	21,716	21,667	22,841	21,716

Table 6: Comparison of the amount of memory used and the volume of traffic reports generated by Flow Slices and ANF for different slice lengths (in Flow Slices) and bin sizes (in ANF). The $t_{inactive}$ columns show the memory savings Flow Slices achieve by using an inactivity timeout of 15 seconds.



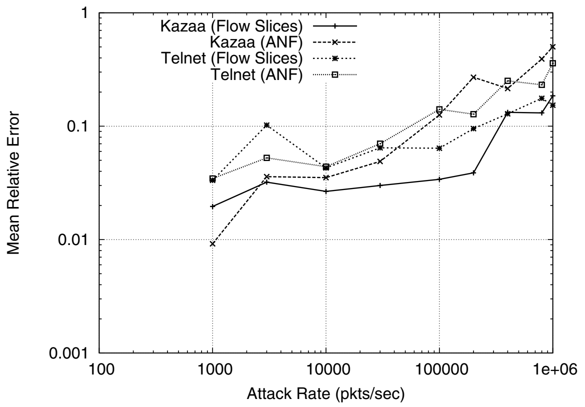


Figure 5: Accuracy of ANF and Flow Slices the rate of attack traffic increases. The left plot compares the average relative error in estimating the size of flows with more than 5,000 packets. The right plot compares the average relative error in estimating the size of two traffic aggregates – telnet and Kazaa. All results are averages over 10 runs with different seeds.

inactivity timeout, the memory usage of Flow Slices is less sensitive to the slice length. The lower memory usage of Flow Slices compared to ANF has important consequences when the sampling rates are adapted dynamically. Given the same memory constraints, the sampling rate adaptation algorithm can converge to more aggressive sampling rates for Flow Slices which results in more accurate estimates.

What is the effect of Denial-of-Service attacks? Figure 5 compares the estimates obtained by ANF and Flow Slices in the presence of a DoS attack. We varied the attack rate from 1000 packets-per-second (pps) to 1.6 million pps; each attack packet represents a different flow as source addresses are spoofed at random. We configured ANF and Flow Slices to operate within a memory budget of 8,000 flow records (not including the buffering needed by ANF to transmit the records at the end of the measurement bin). ANF converged to smaller sampling probabilities as attack traffic gained intensity; the sampling probability varied from 0.155% at 1,000 pps to 0.0026% at 1.6 million pps. Similarly, for Flow Slices, while the random packet sampling probability remained constant at $q = 1/16$ (to simulate real hardware constraints), the combined sampling probability $p{\cdot}q$ varied from 0.781% to 0.0156%. Flow Slices could afford more aggressive sampling mainly due to the use of an inactivity timeout of 15 seconds (the slice length for Flow Slices and bin size for ANF were 60 seconds). On the left, we plot the attack rate on the x-axis and the mean relative error (both for packet and byte counts) of flows with more than 5,000 packets on the y-axis, both in log-scale. For comparable memory usage, in the presence of DoS attacks, Flow Slices produces traffic estimates an order of magnitude better than those of ANF. On the right, we plot the average relative error in estimating traffic that belongs to two different applications – telnet and Kazaa, using the two flow measurement solutions. While the accuracy of both the estimates reduces as the attack rate increases, Flow Slices provides better accuracy than ANF.

While these results do not prove that for all traffic mixes, Flow Slices perform better than other solutions, these results do show the efficacy of the Flow Slices on realistic traffic mixes. When we apply inactivity timeouts to the Flow Slices, it results in much better re-use of memory at the cost of a small loss in accuracy and a little increase in the total volume of flow records reported.

7 Conclusions and future work

Processing, memory, and bandwidth constraints make it impossible for high speed routers to provide full flow measurements, thus forcing us to consider some type of data reduction. Different flow measurement solutions perform this data reduction differently, and one can compare them by comparing their resource consumption and the amount of error the data reduction causes in various analyses one wants to perform on the flow data. Flow Slices offers a unique mix of qualities among flow measurement solutions: dynamic adaptation of sampling parameters to keep resource usage within limits, separate parameters for controlling the three potential resource bottlenecks, efficient use of available resources, and algorithmic solutions for minimizing the errors introduced by the data reduction. These qualities are possible due to novel algorithms such as the core flow slicing algorithm and multi-factor smart sampling and various new estimators. Our experiments also confirm that compared to the currently used Sampled NetFlow and Adaptive NetFlow, Flow Slices constitutes a better flow measurement solution.

But the fact that Flow Slices supports well the traffic analyses discussed in this paper, does not mean there is no room for improvement. There are many useful analyses of unsampled flow data that we haven't considered. For instance, correlation across flows has been used in [19, 13] to classify different flows (such as control and data connections for the same FTP session) into one application. Additional metrics such as flow duration and the variability of packet inter-arrival times have been used to divide flows into different application categories [21]. Such analyses require additional information to be preserved by sampling techniques in order for them to be effective. It is an important challenge to adapt data reduction techniques such as sampling to enable such sophisticated traffic analyses.

8 Acknowledgments

We wish to thank Nick Duffield for discussions that contributed to the idea of the $\widehat{f}$ estimator and Ken Keys for helping us with the code of Adaptive NetFlow. We also wish to thank the anonymous reviewers whose comments helped greatly improve this paper. This work was made possible through NSF grant ANI0074004.

References

[1] Ipmon - packet trace analysis. `http://ipmon.sprintlabs.com/ packstat/ packetoverview.php`.

[2] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. RFC 2722, Oct. 1999.

[3] N. Brownlee and D. Plonka. IP flow information export (ipfix). IETF working group.

[4] Cooperative association for internet data analysis. `http://www.caida.org/`.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD*, 1998.

[6] C. Cranor, T. Johnson, O. Spatschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, June 2003.

[7] The DAG project. `http://dag.cs.waikato.ac.nz/`.

[8] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *IMW*, Nov. 2001.

[9] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *IMW*, Nov. 2002.

[10] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *Proceedings of the ACM SIGCOMM*, Aug. 2004.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. In *ACM Trans. Comput. Syst.*, Aug. 2003.

[12] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. In *SIGCOMM*, Aug. 2000.

[13] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of p2p traffic. In *IMC*, Oct. 2004.

[14] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *PAM*, Apr. 2001.

[15] R. R. Kompella and C. Estan. The power of slicing in internet flow measurement. Technical report, UCSD, May 2005.

[16] K. McCloghrie and M. T. Rose. Rfc 1213, Mar. 1991.

[17] Cisco NetFlow. `http://www.cisco.com /warp /public /732 /Tech /netflow`.

[18] V. Paxson. Bro: a system for detecting network intruders in real-time. In *Computer Networks*, volume 31, 1999.

[19] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *USENIX LISA*, Dec. 2000.

[20] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA*, 1999.

[21] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification. In *IMC*, Oct. 2004.

[22] Sampled NetFlow. `http://www.cisco.com/ univercd/cc/td/ doc/product/software/ios120/ 120newft/120limit/120s/120s11/ 12s_sanf.htm`.

Poisson versus periodic path probing (or, does PASTA matter?)

Muhammad Mukarram Bin Tariq, Amogh Dhamdhere, Constantinos Dovrolis, Mostafa Ammar
Georgia Institute of Technology
{mtariq,amogh,dovrolis,ammar}@cc.gatech.edu

Abstract— The well-known PASTA ("Poisson Arrivals See Time Averages") property states that, under very general conditions, the fraction of Poisson arrivals that observe an underlying process in a particular state is equal, asymptotically, to the fraction of time the process spends in that state. When applied to network inference, PASTA implies that a Poisson probing stream provides an unbiased estimate of the desired time average. Our objective is to examine the practical significance of the PASTA property in the context of realistic RTT, loss rate and packet pair dispersion measurements with a finite (but not small) number of samples. In particular, we first evaluate the differences between the point estimates (median RTT, loss rate, and median dispersion) that result from Poisson and Periodic probing. Our evaluation is based on a rich set of measurements between 23 PlanetLab hosts. The experimental results show that in almost all measurement sessions the differences between the Poisson and Periodic point estimates are insignificant. In the case of RTT and dispersion measurements, we also used a non-parametric goodness-of-fit test, based on the Kullback-Leibler distance, to evaluate the similarity of the distributions that result from Poisson and Periodic probing. The results show that in more than 90% of the measurements there is no statistically significant difference between the two distributions.

1 Introduction

In the context of active measurements, a sequence of probing packets is injected in a network path with the objective to measure end-to-end properties such as Round-Trip Time (RTT), loss rate, or available bandwidth (related to the time-spacing, or dispersion, between successive packets). A central issue in active measurements is the characteristics of the probing process [1]. From a mathematical perspective, it is often claimed that "the right way" to do probing (or sampling) is to use a Poisson process (meaning that the interarrivals between probing packets should be independent and exponentially distributed with the same mean) [2, 3, 4, 5, 6].

Poisson probing derives its strength from the well-cited *PASTA property*, which stands for "Poisson Arrivals See Time Averages" [7]. Consider a stochastic system and suppose that we want to infer the fraction of time p_X the system spends in a state X. Let us assume that this time average exists. A stream of N "probes" arrives in the system based on a Poisson process, recording the system's state at the arrival time instant. PASTA states that the expected value $E[f_{X,N}]$ of the fraction of Poisson probes that observe the system at state X is equal to the time average p_X. This powerful result does not make any assumptions about the stochastic system, except that the time average p_X exists; stationarity or ergodicity may be needed in order to prove that however.

The underlying reason for the validity of PASTA is that the monitored system cannot "anticipate" the next probing event, due to the memoryless nature of the Poisson measurement process. Notice that the the Poisson probes may be interacting with the monitored system. In the case of a queueing system, for example, probing packets can be backlogged, serviced, or dropped, as any other packet, and they may also affect the magnitude of the time average p_X. It is also important to note that PASTA is an asymptotic result: $f_{X,N}$ tends to p_X as the number of probes N tends to infinity. The sample average $f_{X,N}$, even though an unbiased estimate, can significantly deviate around p_X depending on N and on the variability and correlation structure of the underlying system [8].

On the practical side, the most common measurement approach is to use *Periodic* probing, rather than Poisson probing. For instance, the popular *ping* utility generates a periodic probing stream. An important advantage of Periodic probing is that the duration of a measurement session can be a priori known, given the number of samples and the probing period. Furthermore, scheduling periodic packet transmissions at mainstream (not real-time) operating systems is easier and more accurate than scheduling random, and potentially very short or very long, packet interarrivals. It should be noted that the use of Periodic probing does not mean that the resulting estimate will be necessarily biased, especially if the probing rate is sufficiently high. It is true, however, that only Poisson probing can provably result in an unbiased estimate under general conditions.

An important question, which is still unresolved to the best of our knowledge, is whether PASTA "matters" in the pragmatic context of RTT, loss rate, and bandwidth (or dispersion) measurements in the Internet. Given that Periodic probing has some important practical benefits, we need to know whether Poisson and Periodic probing sequences lead to significantly different estimates. In this paper, we focus on three significant path performance metrics: RTT, loss rate and dispersion of back-to-back packet pairs. Our objective is to evaluate the differences that result from Poisson and Periodic probing, both in terms of point estimates for the first moment of the underlying distributions, as well as in terms of differences in the distributions themselves. Note that, since this is a measurement study over wide-area Internet paths, we do not know whether Poisson and/or Periodic probing measure the actual time average of the previous performance metrics. Instead, we can only examine whether the two probing techniques observe the same (but potentially wrong!) path performance.

In Section 2, we describe our measurement collection process. In Sections 3, 4, and 5 we analyze the RTT, loss rate, and dispersion measurements, respectively. Our results show that in almost all measurement sessions the differences between Poisson and Periodic point estimates are insignificant. In the case of RTT and dispersion measurements, there is no statistically significant difference between the Poisson and Periodic distributions in about 90% of the measurements. We conclude with some additional remarks in Section 6.

2 Measurement methodology

We collected measurements of RTT, packet loss rate, and packet pair dispersion in network paths between 23 PlanetLab hosts. Specifically, we used PlanetLab nodes at the following sites: RNP (Brazil), UC San Diego, UC Berkeley, U-Oregon, U-British Columbia, U-Texas, UIUC, Georgia Tech, CMU, MIT, U-Maryland, INRIA (France), Intel Research Cambridge (UK), UPC (Spain), U-Reykjavik (Iceland), EPFL (Switzerland), Helsinki Inst. of Tech. (Finland), U-Tsinghua (China), Hong Kong Univ. of Science & Tech, National Taiwan Univ, Equinix (Singapore), U-Tokyo (Japan), and U-Melbourne (Australia). The measurements reported in this paper were collected from 53 source-destination pairs. Each *measurement session* (or simply "session") consists of 600 Poisson probes and 600 Periodic probes transmitted at the same average rate. The two probing streams of a session start at the same time, and so they cover approximately the same time window.

A session is characterized by a source-destination pair, the average probing interarrival I, and the packet size L. The probing interarrival was 10ms, 20ms, 50ms, 100ms, 500ms, or 1sec, meaning that a session lasts from 6 seconds to 10 minutes, depending on I. For the RTT and loss rate sessions L was 32, 64, 480, or 1400 bytes, while for the dispersion measurements L was 480, 800, or 1400 bytes. The number of successfully completed sessions was 1272 for each of the RTT and loss rate measurements and 954 for the dispersion measurements.

It turned out that some paths were either too slow for our probing streams (especially with I=10ms or 20ms and large packet sizes), or they were extremely congested. In a pre-processing step, we filtered out all sessions in which the loss rate was higher than 10% in the RTT and dispersion analysis; those sessions were included, however, in certain parts of the loss rate analysis (as noted in § 4). The number of sessions we were left with was 892 for RTT and loss rate and 749 for dispersion. The RTT and dispersion time measurements were obtained with kernel-level (*libpcap*) timestamps, reported in a resolution of one microsecond.

3 RTT measurements

Comparison of median RTTs: The sample median is a robust point estimate for the first moment of a distribution. Let T_e and T_p be the median RTTs estimated from the Poisson and Periodic probing streams, respectively, in a particular session. We define the *relative difference between the two RTT medians* as $\epsilon_T = 2\frac{T_e - T_p}{T_e + T_p}$. Figure 1 shows the empirical CDF of ϵ_T. About 60% of the sessions do not see any difference ($\epsilon_T \approx 0$), while about 85% of the sessions have $|\epsilon_T| < 1\%$. The maximum relative difference is 4%. The results are actually quite similar for the relative difference of RTT means. Consequently, at least in relative error terms, Poisson and Periodic probing result in practically the same RTT estimate. The fact that the two probing processes measure almost equal median RTTs, however, does not mean that they also observe the same RTT distribution; we examine this issue next.

Goodness-of-fit test: To further explore the differences between Poisson and Periodic probing, we also examine the *RTT distributions* measured by the two probing processes. Let S_e and S_p be the Poisson and Periodic RTT samples, respectively, collected in a particular session. We form the following null hypothesis:

$$H_0 : S_e \text{ and } S_p \text{ follow identical distributions.} \qquad (1)$$

The alternate hypothesis is that there is a statistically significant difference between the two distributions. A

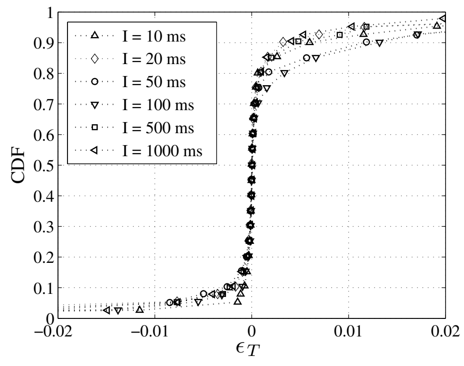
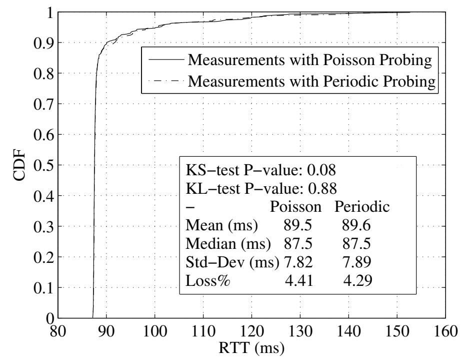


Figure 1: The relative difference ϵ_T between Poisson and Periodic RTT medians.



−	Poisson	Periodic
Mean (ms)	89.5	89.6
Median (ms)	87.5	87.5
Std−Dev (ms)	7.82	7.89
Loss%	4.41	4.29

Figure 2: A session in which the KS test performs poorly.

non-parametric goodness-of-fit test can reject H_0 with a low *P-value* when the two given samples have a statistically significant difference, even if the underlying distributions are strongly non-Gaussian. Recall that P-value is the lowest significance level α at which we can reject the null hypothesis. The P-value is between 0 and 1. H_0 is often rejected when the P-value is less than 0.05-0.10.

Our initial approach was to use the well-known Kolmogorov-Smirnov (KS) test. This test, however, assumes that the underlying distributions are *continuous*. Furthermore, it is quite sensitive to that assumption because it focuses on the maximum vertical distance between the two empirical CDFs. It is important to note that *the RTT distribution at an Internet path can be almost discontinuous* when several probing packets measure the same RTT value that is determined by constant propagation and transmission delays. This discontinuity is located at the minimum (or close to the minimum) measurement, and it is more likely in lightly loaded paths with small queueing delays because the latter can only increase the RTT measurements.

For example, Figure 2 shows the two RTT empirical CDFs measured in a session from MIT to U-Oregon (I=20ms, L=480 bytes). Note that for all practical purposes the two distributions are identical. The KS test, however, rejects H_0 with a low P-value (P=0.08). The reason the test fails is the discontinuity at about 87.5ms. The maximum vertical distance between the two CDFs is 7.4%, it occurs at the 25-th percentile of the Poisson distribution, while the horizontal offset with the Periodic distribution at that point is only 31 microseconds! It is noted that we observed similar failures with other non-parametric statistical tests, such as the Kruskal-Wallis analysis-of-variance test.

To deal with the previous discontinuity problems, we constructed a more robust non-parametric goodness-of-fit test based on the *Kullback-Liebler (KL) distance*, also known as *relative entropy* [9]. For two discrete probability mass functions (pmf's) q_1 and q_2, defined over the same set of values Q, the KL distance of q_1 relative to q_2 is

$$D(q_1 \parallel q_2) = \sum_{i \in Q} q_1(i) \log_2 \frac{q_1(i)}{q_2(i)} \tag{2}$$

It can be shown that $D(q_1 \parallel q_2) \geq 0$ and that $D(q_1 \parallel q_2) = 0$ if and only if the two distributions are identical. Notice that $D(q_1 \parallel q_2) \neq D(q_2 \parallel q_1)$.

The *KL test* proceeds in three steps:

1. Estimate the pmf's s_e and s_p (defined on the same set of bins) from the samples S_e and S_p, respectively. The details of the binning procedure are described in the Appendix.
2. Calculate the KL distance $D(s_e \parallel s_p)$ of the Poisson relative to the Periodic sample.
3. Estimate the distribution of the KL distance $D(s_{e,i} \parallel \bar{s}_{e,i})$ between randomly chosen partitions $S_{e,i}$ and $\bar{S}_{e,i}$ of the Poisson sample S_e ("bootstrapping"). Specifically, suppose that we randomly partition S_e in two samples $S_{e,i}$ and $\bar{S}_{e,i}$. Let $s_{e,i}$ and $\bar{s}_{e,i}$ be the corresponding pmf's, and so $D(s_{e,i} \parallel \bar{s}_{e,i})$ is the KL distance of this partition. If we repeat this random partitioning process many times, we can estimate the distribution of $D(s_{e,i} \parallel \bar{s}_{e,i})$.
4. Reject the null hypothesis if $D(s_e \parallel s_p)$ is "too large" relative to the distribution $D(s_{e,i} \parallel \bar{s}_{e,i})$. Specifically, estimate the P-value as

$$P \approx \text{Prob}[D(s_e \parallel s_p) \leq D(s_{e,i} \parallel \bar{s}_{e,i})] \tag{3}$$

and reject H_0 if $P < 0.1$.

Figure 3 shows an example of the distribution of $D(s_{e,i} \parallel \bar{s}_{e,i})$ together with the KL distance $D(s_e \parallel s_p)$ for a particular session. The KL test is more robust than the KS test in the presence of CDF discontinuities. The reason is that instead of relying on a single point of maximum vertical difference, the KL test considers the difference between the two distributions across all bins, weighted by the probability mass at each bin. For the example of

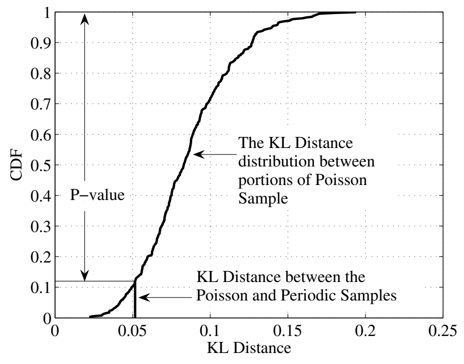


Figure 3: An example of the KL distance distribution for a particular session.

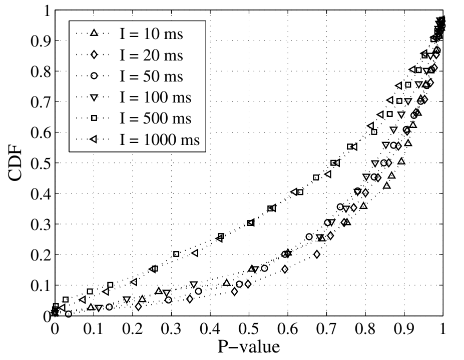


Figure 4: Distribution of P-values for the null hypothesis H_0, applied to RTT sessions.

Figure 2, the KL test reports a P-value of 0.88, meaning that the null hypothesis cannot be rejected.

Figure 4 shows the distribution of P-values reported by the KL test for various probing interarrivals I. We see that we can reject H_0 for only 5-10% of the sessions at a significance level of 10%. The rest of the sessions have significantly higher P-values, meaning that we cannot reject H_0. So, *for more than 90% of the measurement sessions we can assume that the RTT distributions observed by Poisson and Periodic probing are identical.*

Note that the two larger average probing periods, 500ms and 1000ms, result in lower P-values, implying wider (but still not statistically significant) differences between the two RTT distributions. When I is 500ms and 1000ms the measurement process lasts for 300sec and 600sec, respectively, and the underlying process shows significant variability during that time window. Consequently, the two probing processes are more likely to observe different values of the underlying process.

4 Loss rate measurements

For the loss rate estimates, we used the same set of sessions as in the case of RTTs, except that we now use only the 600 probing packets from the source to the destination (ignoring the packets in the reverse direction). Let l_e and l_p be the loss rates, defined as the fraction of lost packets from the Poisson and Periodic streams respectively, in a given session. The *difference between the two loss rates* is $\epsilon_l = l_e - l_p$. Figure 5 shows the empirical CDF of ϵ_l, for the sessions in which we observed some loss, either in Poisson probes, or Periodic probes, i.e., for sessions where $(l_e + l_p) > 0$, but neither l_e or l_p exceeds 10%. Note that in about 80% of the sessions the two loss rates are within 1%, while the maximum loss rate difference is less than 3%.

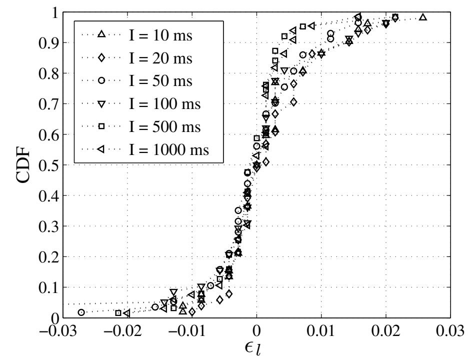


Figure 5: The difference ϵ_l between Poisson and Periodic loss rate estimates.

In theory, we could compare the two loss rates l_e and l_p using a hypothesis test for the equality of two proportions. Such tests however assume that the underlying loss events are independent, which is not true for Internet losses. Instead, we examine the agreement between Poisson and Periodic probing categorically, classifying the sessions in six classes depending on the loss rate estimated by the Poisson probes. These categories are: *lossless* ($l_e = 0$), *low loss* ($l_e \in (0, 1\%]$), *medium loss* ($l_e \in [0.9\%, 5\%]$), *high loss* ($l_e \in [4.5\%, 10\%]$), *very high loss* ($l_e \in [9\%, 20\%]$), and *broken* ($l_e \in [18\%, 100\%]$). The categories have a small overlap to avoid boundary effects when examining the agreement between the two probing processes. Note that for this categorization and in the following analysis we include the sessions that have a higher loss rate than 10%.

Figure 6 shows the fraction of sessions in each category (the number at the top of each group). About 78% of the sessions are classified as lossless or low loss. For each category we also show, with two or three adjacent bars, the fraction of sessions for which the loss rate estimates with Poisson and Periodic probing agree (central bar), as well as the fraction of sessions for which Periodic probing leads to a lower (left bar) or higher (right bar) category. Notice that the two probing techniques agree in more than 70% of the sessions. Given that the loss process in a congested link can be highly bursty,

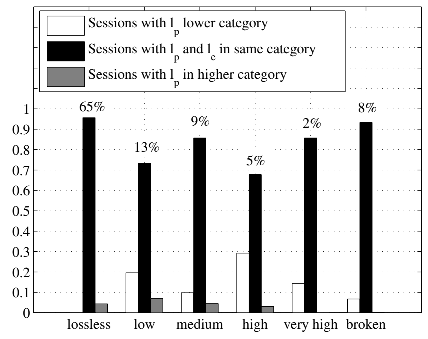


Figure 6: Classification of loss rate estimates in six categories.

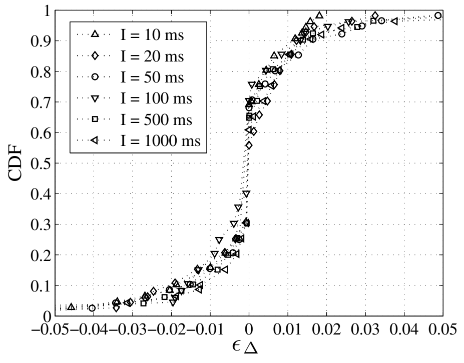


Figure 7: The relative difference ϵ_Δ between Poisson and Periodic dispersion medians.

and since our sample size is rather small to accurately estimate low loss rates, it is not surprising that about 10-30% of the sessions observe different loss categories with Poisson and Periodic probing.

5 Packet pair dispersion measurements

Comparison of median dispersions:

In the dispersion measurements, we send back-to-back packet pairs of size L from the source to the destination. The latter measures the time spacing ("dispersion") Δ between the arrival of the first and the second packet. The dispersion at the destination is related to the cross traffic load and available bandwidth in the network path [4]. Specifically, the higher the cross traffic load is at the path's bottleneck, the wider the dispersion of the packet pairs at the destination due to the interference of cross traffic between the probing pair.

For these measurements, if one of the two packets is lost, the corresponding pair is ignored. Note that the average probing interarrival I controls the time spacing between successive packet pairs, not between packets of the same pair, which are always sent back-to-back.

Let Δ_e and Δ_p be the median dispersions estimated

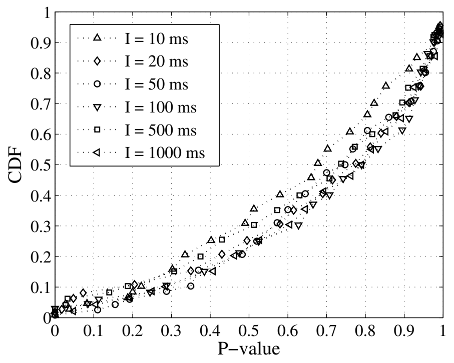


Figure 8: Distribution of P-values for the null hypothesis H_0, applied to dispersion sessions.

from the Poisson and Periodic probing streams, respectively, for a particular session. We define the *relative difference between the two dispersion medians* as $\epsilon_\Delta = 2\frac{\Delta_e - \Delta_p}{\Delta_e + \Delta_p}$. Figure 7 shows the empirical CDF of ϵ_Δ. Note that about 90% of the sessions have $|\epsilon_\Delta| <$2.5%, while the relative difference does not exceed 8%. Consequently, as in the case of RTTs, *we see that the Poisson and Periodic probing processes estimate practically the same dispersion*, at least in terms of a point estimate for the first moment.

Goodness-of-fit test: We also examined whether the two probing processes observe the same dispersion distribution. To do so, we used again the KL test described in § 3. This time the null hypothesis H_0 is that the two dispersion samples (Poisson and Periodic) for a given session follow an identical distribution. Figure 8 shows the distribution of P-values reported by the KL test, for various probing interarrivals I. Notice that the P-value is less than 10% for only 5-10% of the sessions. The rest of the sessions have significantly higher P-values, implying that we can assume H_0 to be true.

6 Discussion

The experimental results in this paper indicate that there may not be a significant difference between Poisson and Periodic probing, at least in the context of real Internet measurements. *This does not mean that we recommend the use of Periodic probing over Poisson probing.* We note however that measurement studies that use, or have used, Periodic probing should not be dismissed based on that fact, and they may also have practical benefits compared to Poisson probing. A few additional remarks on the accuracy of Poisson and Periodic probing follow.

First, it is important to note that the fraction of dropped Poisson probing packets at a network queue does *not* estimate the packet loss rate, i.e., the fraction of dropped packets among *all* arrived packets; instead, it estimates *the fraction of time that the queue is full.* The latter is equal to the loss rate in a queue with Poisson packet arrivals; this follows applying the PASTA property to all packets (not only the probing packets). For more bursty traffic, however, the packet loss rate can be higher than the fraction of time that the queue is full. Consequently, even if Poisson and Periodic probing observe the same loss rate, that fraction should not be expected to be equal to the underlying loss rate.

Second, Periodic probing at a certain interarrival I cannot "see" effects that occur in lower time scales (or higher frequency). In the case of loss rate estimation, in particular, loss events in Drop-Tail queues can be very bursty. If the duration of loss bursts is much lower than I, then Periodic probing may underestimate both the full-queue probability and the loss rate. Poisson probing, however, with a sufficiently large number of samples, should be able to estimate the full-queue probability.

Third, it is important to note that even if we generate a Poisson probing stream at the source, the probing packets may not arrive at the bottleneck link as a Poisson process. Consider, for instance, that the output of an $M/D/1$ queue is *not* a Poisson process. In more practical terms, if the probing packets go through a store-and-forward link with capacity C, then their interarrivals after that link cannot be lower than L/C, where L is the packet size. Consequently, the probing packets will no longer be a Poisson stream and PASTA will not apply. This issue is important for Internet measurements, given that most network paths go through multiple queues.

Appendix: Density estimation

Suppose that we are given two samples S_1 and S_2 that take values in a range R_S. We want to approximate the probability density functions of the two samples with the probability mass functions $q_1(i)$ and $q_2(i)$ defined over a set of bins Q, such that $q_1(i)$ and $q_2(i)$ is the fraction of measurements in the $i'th$ bin from S_1 and S_2, respectively, and $\sum_{i \in Q} q_1(i)$=$\sum_{i \in Q} q_2(i)$=1. We select the initial bin size w based on the Freedman-Diaconis rule as $w = 2n^{-1/3}$IQ, where n is the number of samples in the joint sample $S_1 \cup S_2$ and IQ is the interquartile range of R_S [10]. We then proceed to determine the bin boundaries, with the first bin placed based on the minimum measurement, and to estimate the functions $q_1(i)$ and $q_2(i)$.

The problem, however, is that some bins may not include enough measurements from each sample. When that is the case we are not able to accurately estimate the "likelihood ratio" $log_2 \frac{q_1(i)}{q_2(i)}$ of Equation 2. To guarantee that each bin contains at least a certain number of measurements from both samples, we use an adaptive histogram approach. Specifically, if a bin does not include at least 1% of the measurements from each sample, we merge that bin with the bin at its right. The process is repeated until the previous constraint is met. If there are no measurements from that sample at any of the right bins, we merge the problematic bin with the bin at its left. In practice, we found that the previous heuristic is quite robust, as long as the two samples take values over approximately the same range.

Acknowledgments

We are grateful to Prof. Paul Kvam (ISYE, Georgia Tech) for suggesting the use of relative entropy in a goodness-of-fit test. This work was supported in part by the NSF CAREER award ANIR-0347374, NSF award ANI-024085, and by a gift from DoCoMo USA Labs.

References

[1] V. Paxson, G.Almes, J.Madhavi, and M.Mathis, *Framework for IP Performance Metrics*, May 1998, RFC 2330.

[2] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*, Ph.D. thesis, University of California, Berkeley, Apr. 1997.

[3] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," in *Proceedings of ACM/USENIX Internet Measurement Workshop (IMW)*, Nov. 2001, pp. 197–211.

[4] J. Strauss, D. Katabi, and F. Kaashoek, "A Measurement Study of Available Bandwidth Estimation Tools," in *Proceedings of ACM/USENIX Internet Measurement Conference (IMC)*, 2003.

[5] D. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Analysis of Measured Single-Hop Delay from an Operational Backbone Network," in *Proceedings of IEEE INFOCOM*, 2002.

[6] R. Caceres, N. G. Duffield, J. Horowitz, and D. Towsley, "Multicast-Based Inference of Network-Internal Loss Characteristics," *IEEE Transactions in Information Theory*, pp. 2462–2480, 1999.

[7] R. Wolff, "Poisson Arrivals See Time Averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.

[8] J. Sommers, P. Barford, N. G. Duffield, and A. Ron, "Improving Accuracy in End-to-End Packet Loss Measurements," in *Proceedings ACM SIGCOMM*, Aug. 2005.

[9] M. Lexa, "Useful Facts about the Kullback-Leibler Discrimination Distance," Tech. Rep., Rice University, 2004.

[10] D. Scott, *Multivariate Density Estimation: Theory, Practice and Visualization*, Prentice Hall, 1992.

On the Accuracy of Embeddings for Internet Coordinate Systems

Eng Keong Lua, Timothy Griffin, Marcelo Pias, Han Zheng, Jon Crowcroft
University of Cambridge, Computer Laboratory
Email: {eng.keong-lua, timothy.griffin, marcelo.pias, han.zheng, jon.crowcroft}@cl.cam.ac.uk

Abstract

Internet coordinate systems embed Round-Trip-Times (RTTs) between Internet nodes into some geometric space so that unmeasured RTTs can be estimated using distance computation in that space. If accurate, such techniques would allow us to predict Internet RTTs without extensive measurements. The published techniques appear to work very well when accuracy is measured using metrics such as *absolute relative error*. Our main observation is that absolute relative error tells us very little about the *quality* of an embedding as experienced by a user. We define several new accuracy metrics that attempt to quantify various aspects of user-oriented *quality*. Evaluation of current Internet coordinate systems using our new metrics indicates that their quality is not as high as that suggested by the use of absolute relative error.

1 Introduction and Motivation

An Internet coordinate system starts with a collection of nodes and measured Round-Trip-Times (RTTs) between some pairs of these nodes. It then *embeds* the nodes into a geometric space by associating each node with a point in that space. The goals of such an embedding are twofold. First, the embedding should be *accurate* in the sense that the geometric distance between embedded nodes should, in some sense, closely approximate their RTTs. Second, the methods should remain accurate even when the input is a small subset of all possible RTT measurements. That is, a node's coordinates should allow even the unmeasured RTTs to be estimated accurately.

Many Internet coordinate systems have been described in the literature. Accuracy is typically assessed using metrics similar to *absolute relative error*, which forms an average over every pair of nodes of the absolute value of the difference between their embedded distance and their original distance. Note that an embedding technique does not require a full mesh of RTT measurements, but assessing its accuracy surely does. The main conclusion of this paper is that by itself an accuracy metric such as absolute relative error tells us very little about the *quality* of an embedding as experienced by a user.

Our own experiences and experiments with Internet coordinate systems have been disappointing in several respects. First, distance estimation results are often *unpredictable* in the sense that many nodes obtain good estimates while a few obtain extremely bad results. In a real-world setting it seems that nodes cannot determine the *quality* of their estimates without performing exactly the full probing that coordinate systems are intended to eliminate. Second, we observed that the *quality* of an embedding often varied considerably with small changes to the topology of the underlying network — something which would be beyond the control of most users of a coordinate system. Third, we observed that the *quality* of embeddings often varies considerably as the number of participating nodes changed, even when the underlying topology remains fixed. In short, these observations suggest that it is very hard to predict when a coordinate system will work well at any given node.

Highly aggregated accuracy metrics, such as absolute relative error, seem to give little indication of these types of *quality* problems. This experience led us to question the usefulness of metrics such as absolute relative error, at least in isolation from other means of assessing *quality*. This paper presents several new accuracy metrics that attempt to capture more application-centric notions of *quality*. The first is called *relative rank loss at node A* (rrl at node A), and is intended to be useful for applications used by nodes that need to know only their *relative* distance of other nodes. That is, node A needs to answer question such as "Which node is closer to me, B or C?" — which we will call A's proximity query for nodes B and C. The metric rrl at node A is defined as a type of *swap distance* and takes values between 0 (all proximity queries at A are correctly estimated) and 1 (none of the proximity queries at A are correctly estimated). For example, an rrl at node A of 0.5 tells us that node A has a 50 percent chance of getting the correct answer to a proximity query. We then define aggregate versions of rrl such as the average rrl over all nodes, and the maximal rrl value over all nodes. The other new metric we define is *closest neighbors loss at node A* (cnl at node A). This metric is intended to be useful to nodes using applications that are interested only in determining which nodes are *closest*. The value of cnl at node A is 0 if the nodes closest to A remain closest in the embedding, and 1 otherwise. Again, we then define aggregate versions of cnl such as the average global cnl over all nodes. For example, an average global cnl value of 0.5 would tell us that 50 percent of the nodes have a different set of closest neighbors in the original space than they have in the embedded space.

When we compare the accuracy of embedding techniques using our new metrics the results can be very poor, even in simple tree and hub-and-spoke networks. Our new metrics capture and quantify, at least partially, some of our disappointment with the *quality* of the embedding techniques. However, it remains that no single accuracy metric can capture the *quality* of an embedding technique. It seems better to develop a collection of accuracy metrics that are each designed to quantify a specific aspect of user-oriented *quality*. By no means do we want to suggest that our new metrics represent a complete or definitive set — they are simply an initial attempt at exploring this space.

This paper is complementary to the work presented in [23], which shows that RTT violations of the triangle inequality are a common, persistent, and widespread consequence of Internet routing policies. We strongly suspect that such violations will degrade *every* embedding technique with respect to *any* accuracy metric, at least when the embedding maps into a geometric space where the triangle inequality does hold. The results of the current paper and those of [23] are not very encouraging, to say the least. On the positive side, we feel that research is always improved with a better understanding of the fundamentals. In this case, we believe an improved understanding suggests a new and potentially interesting line of research, which we will call Embeddable Overlay Networks (EONs). An EON is an overlay where routing nodes have been *selected* to avoid violations of the triangle inequality (for overlay forwarding) *and* the overlay topology has been *selected* to embed with high accuracy with respect to multiple useful accuracy metrics. To do this well will require new insights and new algorithms.

Paper Outline. Section 2 presents a short survey of the embedding techniques described in the literature. Section 3 presents the basic terminology of network embeddings and our new accuracy metrics. In Section 4 we present an embedding experiment using our data collected on PlanetLab. We use a *full* Lipschitz embedding technique which makes use of all nodes as the landmarks (reference nodes) without any dimensionality reduction, and evaluate the results with our new metrics. In Section 5, we conducted experiments using our PlanetLab data sets and new accuracy metrics, but on other techniques that are based on numerical minimization — Vivaldi [8] and Big Bang Simulation (BBS) [18–20]. In Section 6, we revisit these previously published experiments and use their data sets to re-evaluate their accuracy with our new accuracy metrics. Section 7 concludes with some topics for future work.

2 A Brief Survey of Embedding Techniques

Among the recently described Internet coordinate systems are the Global Network Positioning (GNP) [16], Lighthouses [17], Virtual Landmarks [21], Internet Coordinate System (ICS) [11], Matrix Factorization [15], Practical Internet Coordinates (PIC) [7], Vivaldi [8], Big Bang Simulation (BBS) in Euclidean space [18], and BBS in Hyperbolic space [19, 20]. Basically, Internet coordinate systems comprise of *two* categories of network embeddings of finite metric space into low dimensional Euclidean or non-Euclidean (e.g. Hyperbolic) space, namely, *numerical minimization* of some defined objective distance error functions for nodes-to-landmarks distances; and initial Lipschitz embedding with some form of dimensionality reduction using *distance matrix factorization* (which is a form of mathematical optimization and minimization techniques of error functions).

ICS and Virtual Landmarks systems are based on Lipschitz embedding of a finite metric space into Euclidean space and use the Principal Component Analysis (PCA) to reduce dimensionality. Typically, the accuracy of such embeddings is studied *only after* some type of dimensionality reduction technique has been applied. Matrix factorization based on Singular Value Decomposition (SVD) is of the form $D = USV^T$, where D is the distance matrix, U and V are orthogonal matrices and S is an diagonal matrix with nonnegative elements arranged in decreasing order which measure the significance of the contribution from each principal component. This is related to PCA on the distance matrix row vectors in ICS and Virtual Landmarks, where the first d rows of the matrix U are used as coordinates for the network nodes. The Matrix Factorization embedding uses the distance matrix whose rows are not linearly independent (has rank strictly less than n where n is the number of nodes). It is expressed as the product of two smaller matrices of A and B (through SVD or Nonnegative Matrix Factorization (NMF) algorithms), which contain the outgoing and incoming vectors respectively for each node. Estimated distances between nodes are derived from the dot product of these *two* vectors. Lighthouses technique makes use of multiple local bases L together with a transition matrix in vector space, to allow flexibility for any node to determine coordinates relative to any set of landmarks provided it maintains a transition matrix for global basis G. It uses linear matrix factorization such as the QR

decomposition (Gram-Schmidt orthogonalization).

For techniques involving minimization of some defined objective distance error functions, many algorithms are proposed to compute the coordinates of the network nodes. For example, GNP and PIC systems use the Simplex Downhill to minimize the objective distance error function: sum of relative errors. The problems of using the Simplex Downhill method is the slow convergence, sensitive to the *initial* coordinates or position of the network nodes, and the potential of getting stuck in the local minima. This leads to the eventual assignment of different coordinates for the same node depending on the minimization process (e.g. selection of the initial position). So, methods that are clever to converge to the global minimum are required. Both Vivaldi and BBS (Euclidean and Hyperbolic spaces) algorithms are based on the numerical minimization of sum of distance error functions that are related to the problem of minimizing the potential energy of newtonian mechanics principles. Vivaldi system is constantly adjusting the coordinates of the nodes because each node contacts a random set of nodes in a decentralized manner. Each node cannot find a global minimum fit, and it cannot guarantee that a given error function minimization does not increase the global error, although Vivaldi ensures nodes always decrease the local error. Hence, the error term in each small time-step is monitored to derive the optimal sets of coordinates selected.

The Big-Bang Simulation (BBS) (Euclidean and Hyperbolic) systems model the network nodes as a set of particles, having a position in Euclidean space. The nodes are traveling under the effect of the potential force field which reduces the potential energy of the nodes. This is related to the total embedding distance error of all node pairs in Euclidean space. Each node pair is effected by the field force induced between them depending on their node pair's embedding distance error, i.e. the embedding distance error of the distance between the node pair. The node is also affected by simulated friction force. BBS systems have the advantage over conventional gradient minimization schemes, such as steepest descent and Simplex Downhill, due to the kinetic energy accumulated by the moving nodes which enable them to escape the local minima. Hyperbolic geometric space is the target embedding space for the BBS (Hyperbolic) system. A distance decreases as it moves away from the origin. Similar to Euclidean line distance, the Hyperbolic distance line between two nodes is defined as the parametric curve, connecting between the nodes, over which the integral of the arc length is minimized. Unlike Euclidean line distance, a Hyperbolic line distance bents towards the origin point. The extent to which the bend depends on the curvature of the Hyperbolic space. The bending becomes larger when space curvature increases, and thus, Hyperbolic distance between two nodes increases. There are three embedding methods in BBS systems: **All Pairs (AP) embedding** — Embedding is done for *full* mesh of the n nodes of the network topology comprising of $\frac{n(n-1)}{2}$ distance pairs (n is the number of network nodes). **Two Phases (TP) embedding** — Embedding is done using landmarks similar to GNP, where the landmarks are embedded first and the coordinates of the other nodes are calculated from the distance to *several* chosen *closest* landmarks through minimization of the symmetric distortion in these node-to-landmark pairs. Specifically, TP embedding requires $k + 1$ landmarks' measurements for k-dimensional coordinate vectors. **Log-Random and Neighbors (LRN) embedding** [20] — It aims to increase neighbor distances accuracy. The LRN embedding concurrently embed nodes through minimization of objective error function of n nodes and LRN subset, which comprises of the node pairs whose distance is below a certain threshold, i.e. the threshold is selected so that the number of distance pairs that are below the threshold is $O(n. \log\ n)$, and together with a set of randomly sampled distance pairs that are selected uniformly at random with probability $\frac{\log\ n}{n}$. The number of randomly sampled distance pairs is equivalent to $n. \log\ n$. The LRN algorithm embeds all the n nodes concurrently. The objective function is the sum of embedding errors for all n nodes in the system, and the embedding error of one node is the error of distance from that node to the LRN subset of nodes.

3 Embeddings and Their Accuracy

This section rigourously defines several accuracy metrics for embeddings. We use the Lipschitz embedding as a running example to illustrate and motivate our metric definitions.

A *metric space* is a pair $M = (X,\ d)$ where X is a set equipped with the distance function $d : X \to \mathbb{R}^+$ — for each $a,\ b \in X$ the *distance between* a *and* b is given by the function $d(a,\ b)$. We require that for all $a,\ b, c \in X$,

(anti-reflexivity) $d(a,\ b) = 0$ if and only if $a = b$,

(symmetry) $d(a,\ b) = d(b,\ a)$,

(triangle inequality) $d(a,\ b) \leq d(a,\ c) + d(c,\ b)$.

Note that we will ignore the fact that in reality, violations of the triangle inequality exist for Internet RTTs ([23]).

Suppose that $M_1 = (X_1,\ d_1)$ and $M_2 = (X_2,\ d_2)$ are metric spaces. Every one-to-one function ϕ from X_1 to X_2 naturally defines a metric space called *the embedding of* M_1 *in* M_2 *under* ϕ, defined as $\phi(M1) = (\phi(X_1),\ d_2)$. We normally abuse terminology and simply say that ϕ embeds X_1 into X_2, leaving the distance functions to be inferred from the context. If ϕ embeds X_1 in X_2 and Z is a subset of X_1, then ϕ also defines an embedding of Z in X_2 in the natural way. One simple case is where X_1 is a finite set,

X_2 is $\mathbb{R}^n$ with the standard notion of Euclidean distance, $d_2(x,\ y) = \|x - y\| = \sqrt{\sum_{1 \le i \le n} (x_i - y_i)^2}$.

If $L = \{l_1, l_2, \ldots, l_m\} \subseteq X$, then we can map each $a \in X$ to a point in $\mathbb{R}^m$ as $\phi_L(a) = (d(a,\ l_1),\ d(a,\ l_2),\ \ldots,\ d(a,\ l_m))$. This is called the *Lipschitz embedding of X using landmarks L*. If $L = X$, we refer to $\phi_X = \phi$ as the *full Lipschitz embedding*.

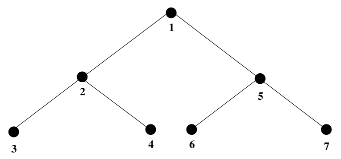


Figure 1: A binary tree of depth 2.

We illustrate the *full* Lipschitz embedding with a simple example. Any weighted undirected graph $G = (V,\ E,\ w)$ naturally gives rise to a metric space $M(G) = (V,\ d)$, where $d(a, b)$ is the length of a shortest path between nodes a and b. Figure 1 presents one example — a simple binary tree of depth 2, where each edge represents a distance of 1, and gives rise to the following distance matrix:

	1	2	3	4	5	6	7
1	0	1	2	2	1	2	2
2	1	0	1	1	2	3	3
3	2	1	0	2	3	4	4
4	2	1	2	0	3	4	4
5	1	2	3	3	0	1	1
6	2	3	4	4	1	0	2
7	2	3	4	4	1	2	0

Note that this distance matrix itself can be viewed as a full Lipschitz embedding into a $\mathbb{R}^7$ by reading each row as the 7-dimensional coordinate of the associated node. For example, $(0,\ 1,\ 2,\ 2,\ 1,\ 2,\ 2)$ are the coordinates of node 1.

3.1 Relative Error

We seek embeddings that are *accurate*. There are several ways to capture this formally, and in the context of Internet coordinate systems the most appropriate notion may depend on the needs of an application. Some applications require that the distances in the embedding accurately reflect the original distances. Note that with the binary tree example above distances are not well preserved — we can calculate the distance between $\phi(1)$ and $\phi(7) = (2,\ 3,\ 4,\ 4,\ 1,\ 2,\ 0)$ as approximately 4.47, yet it is only 2 in the original metric space.

If we multiply each coordinate $\phi(x)$ by a scalar value β, and redefine the embedding as $\phi'(x) = \beta\phi(x)$, we could improve this notion of accuracy. As in [21], we choose β to minimize the *absolute relative error* for all pairs of nodes, given by $\sum_{x,\ y \in X} \frac{|\|\phi'(x) - \phi'(y)\| - d(x,\ y)|}{d(x,y)}$. For the example above, this gives a value for β of approximately 0.5, which reduces the distance between the embeddings of nodes 1 and 7 to approximately 2.3. The *average absolute relative error* is obtained by dividing the absolute relative error with $n(n-1)$, where n is the total number of nodes. The *maximum local absolute relative error* is the maximal *absolute relative error* over all nodes.

A similar accuracy metric is *stress*, defined as $\frac{\sum_{x \neq y \in X} (\|\phi'(x) - \phi'(y)\| - d(x,y))^2}{\sum_{x \neq y \in X} d(x,y)^2}$. Note that both stress and relative error quantify the magnitude of the differences between original distances and embedded distances. If stress (or relative error) is 0, then we have an embedding that preserves distances exactly (an isometry).

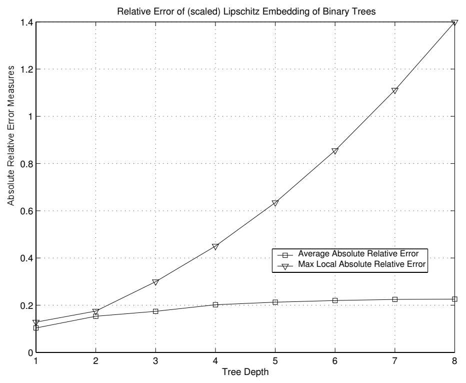


Figure 2: Absolute Relative Error measures for Lipschitz embedding on binary trees, depth 1 to 8.

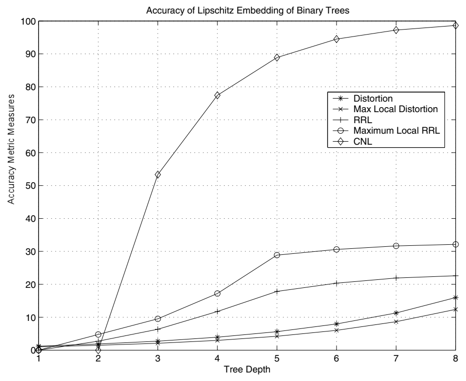


Figure 3: Scalar independent measures for Lipschitz embedding on binary trees, depth 1 to 8.

3.2 Distortion

In the theoretical literature on embeddings (for example, [10, 12–14]) the most common notion of accuracy is captured by *distortion*. This measure is invariant under scalar

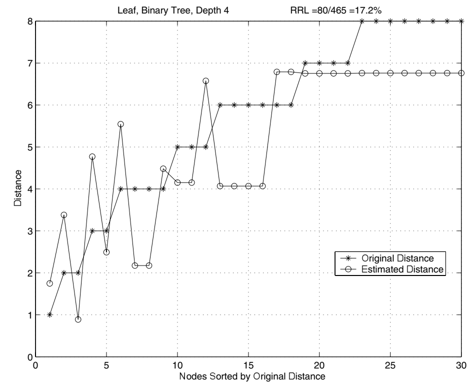


Figure 4: The view from a leaf in a binary tree of depth 4.

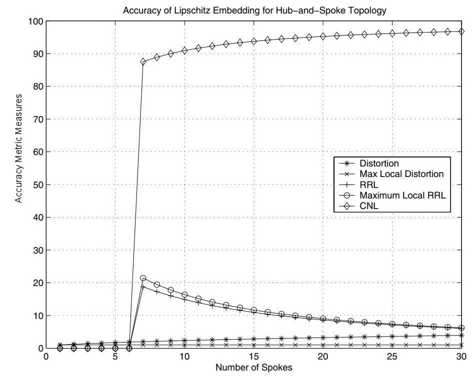


Figure 5: Hub and Spoke accuracy.

multiplication. That is, $distortion(\phi) = distortion(\alpha\phi)$, for all $\alpha \neq 0$ — and so provides a notion of accuracy that is independent of the absolute values of distances. Distortion measures the worst-case change in the *relative distances* of the embedding.

First, define the ratio $r(\phi,\ x,\ y) = \frac{\|\phi(x)-\phi(y)\|}{d(x,\ y)}$. The *expansion of* ϕ, $expansion(\phi)$, is the maximum value of $r(\phi,\ x,\ y)$ as x and y range over X ($x \neq y$). The *contraction of* ϕ, $contraction(\phi)$, is the minimum value of $r(\phi,\ x,\ y)$ as x and y range over X ($x \neq b$). The *distortion of* ϕ is defined as the ratio $distortion(\phi) = \frac{expansion(\phi)}{contraction(\phi)}$. Note that the $distortion(\phi)$ is always greater than or equal to 1. The equality holds only when there exists a constant α such that $r(\phi x,\ y) = \alpha$ for every $x,\ y$. In this case $\phi'(x) = \phi(x)/\alpha$ is an isometry.

One way to gain intuition for the meaning of distortion is to imagine that $contraction(\phi) = 1$ (which could always be achieved with scalar multiplication). In this case there is no contraction in the embedding, only expansion, and the largest expansion is achieved by some pair $x,\ y$, where the distance between x and y in the embedding is $expansion(\phi)$ times the original distance $d(x,\ y)$.

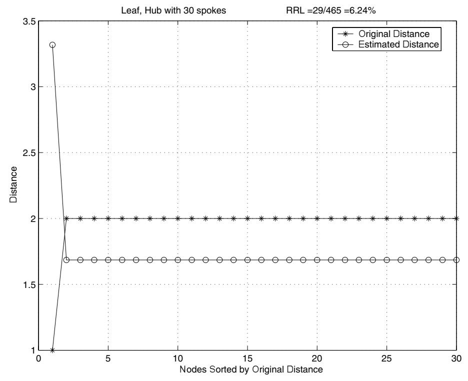


Figure 6: The view from a leaf in a hub with 30 spokes.

Note that distortion is a global worst-case measure, which may be relevant to applications (such as mapping) that require a global picture of the entire embedding. However, most Internet coordinate applications will be interested only in *local distortion* — the amount of distortion seen from any one node. For a fixed x, we define $expansion(\phi,\ x)$, to be the maximum value of $r(\phi,\ x,\ y)$ as y range over X ($x \neq y$), and $contraction(\phi,\ x)$, to be the minimum value of $r(\phi,\ x,\ y)$ as y range over X ($x \neq y$). Then the local distortion is defined to be $distortion(\phi,\ x) = \frac{expansion(\phi,\ x)}{contraction(\phi,\ x)}$. The *maximum local distortion* is the maximal $distortion(\phi,\ x)$ over all x.

3.3 A New Metric: Relative Rank Loss (rrl)

Many applications only need to know the *relative* distance of other nodes. They need to answer question such as "Is A closer than B?" What is important for such applications is that the ***relative rankings** of distances is not lost*. For this notion of accuracy we define *relative rank loss* (rrl) to be between 0 (for no loss of relative order) and 1 (for a complete reversal of order). This is a slight generalization of *swap distance*, a well-known edit distance on strings. For example, between strings $s_1 = abcd$ and $s_2 = acdb$ there is a swap distance of 2. We need to generalize this slightly to account for the fact that multiple nodes can be equidistant from a given node.

First, define the function

$$R(x,\ y) = \begin{cases} \text{-1} & \text{if } x < y \\ 0 & \text{if } x = y \\ 1 & \text{if } x > y \end{cases}$$

We say that *x and y are swapped w.r.t.* z if $R(d(z,\ x),\ d(z, y)) \neq R(\|\phi(z) - \phi(x)\|,\ \|\phi(z) - \phi(y)\|)$ and denote this as $swapped(z,\ x,\ y)$. That is, if $swapped(z,\ x,\ y)$ holds, then z's relative distance relationship to x and y is different in the original and embedded spaces. Define $P(z)$, a subset of $(X - \{z\}) \times (X - \{z\})$, to be

$$\{(x,\ y) \mid x \neq y \text{ and } \mathit{swapped}(z,\ x,\ y)\}.$$

Finally, define the *local relative rank loss at z* to be

$$rrl(\phi,\ z) = \frac{\mid P(z) \mid}{s},$$

where $s = \frac{(|X|-1)(|X|-2)}{2}$. Note that $rrl(\phi,\ z)$ is between 0 and 1. It can be interpreted as the probability that the relationship between any two nodes in $(X - \{z\}) \times (X - \{z\})$ will have a different relative order (from z's point of view) in the original and embedded spaces (assuming all pairs $(x,\ y)$ have equal probability of being chosen).

Define the *maximal local relative rank loss at z* to be the maximal value of $rrl(\phi,\ z)$ as z ranges over X. The *average local relative rank loss at z*, denoted $rrl(\phi)$, is defined as $\frac{\sum_{x \in X} rrl(\phi,\ x)}{|X|}$.

3.4 A New Metric: Closest Neighbors Loss (cnl)

Some applications are interested only in determining which nodes are *closest*, and so require that embeddings accurately preserve the set of closest nodes. For a node x, we define its local *closest neighbors loss* to be 0 if the nodes closest to x in X are mapped to the nodes closest to $\phi(x)$, and 1 otherwise. We denote this value by $cnl(\phi,\ x)$. If we take the average over all nodes in X, we denote the global value as $cnl(\phi)$. We often multiply this global cnl by 100 and speak of the percentage of nodes whose nearest neighbors are not preserved.

3.5 Examples of Inherent Inaccuracy — Topology Matters

We now apply the accuracy measures defined above to several simple topologies. Figure 2 presents the relative error and maximum local relative error measures for binary trees of depth 1 (3 nodes) through 8 (511 nodes). Note that it is not obvious or intuitive how to interpret these (small) values. On the other hand, Figure 3 presents the scalar-independent measures for the same set of binary trees. The plot for cnl tells us that about 96 percent of the 511 nodes in a tree of depth 8 have a different closest neighbors set in the Lipschitz embedding. The plot for rrl shows that on average nodes see over 20 percent of their relative distance relationships swapped. The plot for maximal local rrl tells us that at least one node saw over 30 percent of its relative distance relationships swapped.

Figure 4 helps us to understand these inaccuracies. This plot is from the perspective of a leaf node in a binary tree of depth 4 (31 nodes). The signature plot marked with *****'s shows the distances of other nodes from this leaf, in ascending order. The signature plot marked with **o**'s shows the distances of the same nodes in the Lipschitz embedding. In the original metric space, the parent of this leaf is the only node at distance 1. But in the embedding, this node moves to a distance of about 1.8. There were two nodes originally at distance 2 — the leaf's sibling and its parent's parent. One of these (its sibling) has moved closer (and is now closer than the leaf's parent), to a distance just under 1, while the leaf's parent's parent moves out to a distance of about 3.4. The local rrl for this node is 17.2 percent. Note that it has lost its closest neighbor in the embedding.

Some topologies show a very different relationship between rrl and cnl. For example, Figure 5 shows the scalar-independent measures for a family of hub-and-spoke networks. The graph have n spokes and 1 root, where n ranges from 1 to 30. Here we see a rising cnl and a falling rrl after $n = 6$. The reason becomes clear when we look at the view from a leaf node, as shown in Figure 6. We see that the root nodes is ***pushed away*** to a distance of about 3.3.

3.6 The Scalability (Meta-) Metric

Suppose that a set of applications is only interested in a subset of nodes, say just the North American sites. Would it be better to use a coordinate system generated for all sites, or one generated just from inter-site RTTs restricted to North American sites? The answer to such questions will determine if embeddings of coordinate *services* could be *truly* scalable.

If Y is a subset of X, we can obtain an embedding in one of two ways. First, we could use the embedding techniques to obtain $\phi(X)$, and then restrict this to the nodes in Y, which we will denote as $\phi(X) \downarrow Y$. This is called *Superspace* embedding. Alternatively, we could use the embedding techniques on Y to get $\phi(Y)$, which we call the *Subspace* embedding. In general, $\phi(X) \downarrow Y$ and $\phi(Y)$ may be *very* different embeddings and have *very* different behaviors with different accuracy measures.

For example, consider again Figure 3. Let X be the binary tree of depth 8. We can consider each of the subtrees of depths 1 to 7 as a Subspace Y of X. The figure shows the scalar-independent accuracy measures for each full Lipschitz embedding of Subspace $\phi(Y)$. However, the full Lipschitz embeddings derived from the Superspace, $\phi(X) \downarrow Y$, will in general inherit some of the inaccuracies of the Superspace embedding — that is the nodes outside of Y in X impact the way Y is embedded.

4 An Experiment with the Lipschitz Embedding

Although data sets studied in previous research comprised large number of measurements, some of them were not

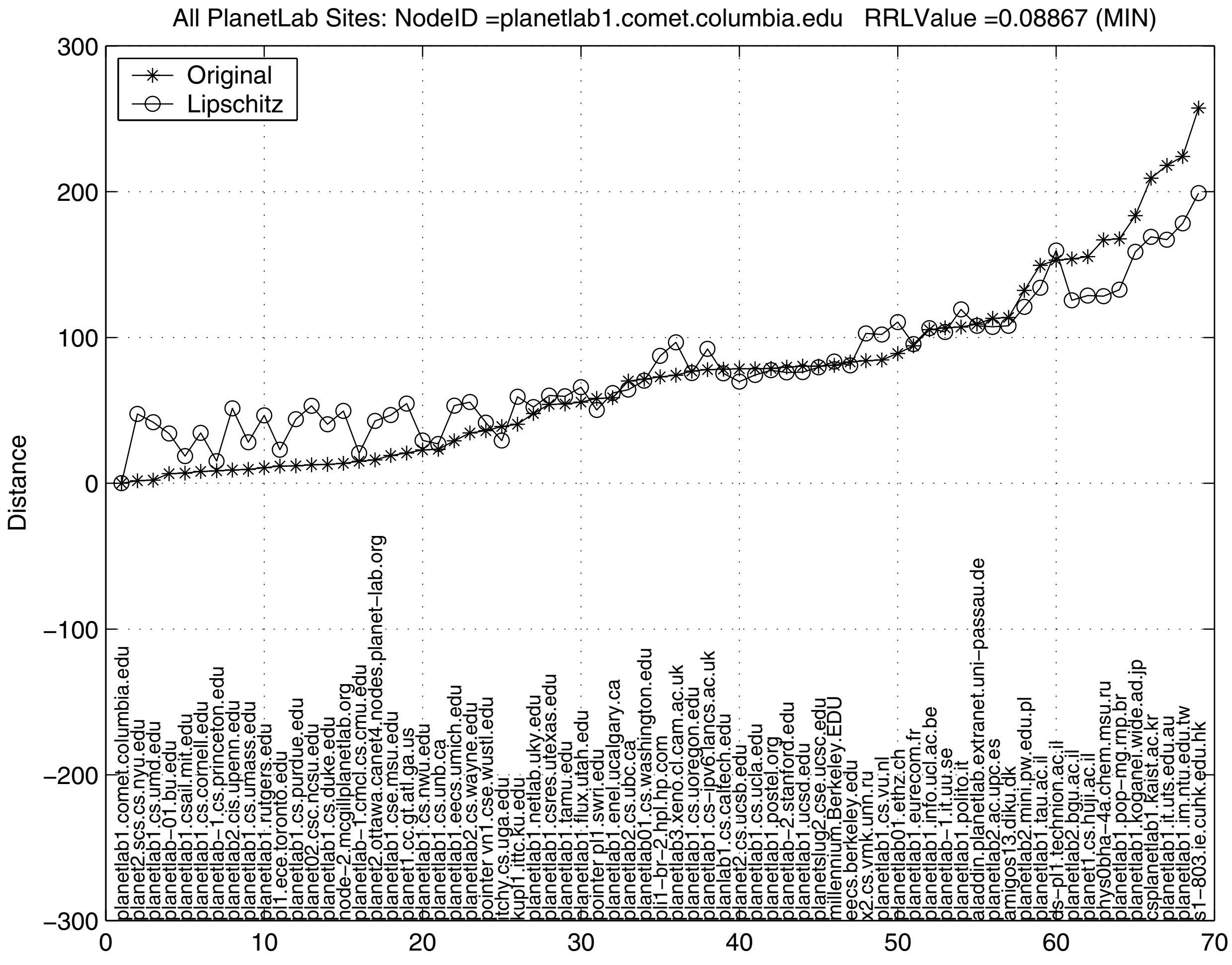


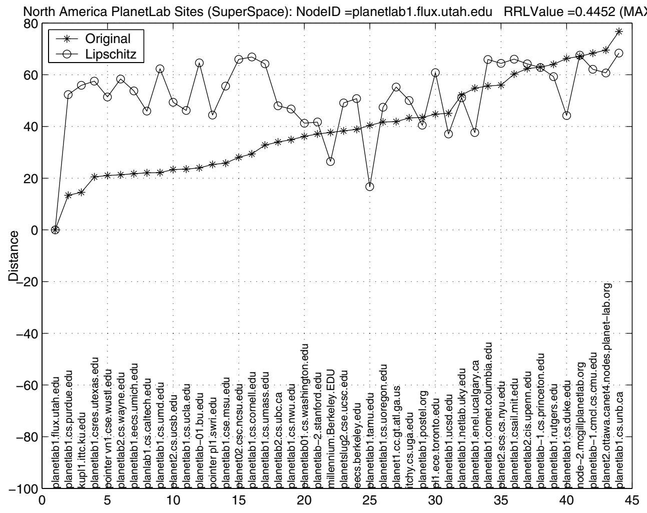
Figure 7: PlanetLab site `comet.columbia.edu` with minimum rrl using *full* Lipschitz embedding.

full meshes. The Skitter project [3] used in Virtual Landmarks [21], for instance, makes RTT data available from a small number of monitoring nodes n to m target nodes, where m is in the order of hundreds of thousands. This yields an asymmetric data set of $n \times m$ where only estimated distances between monitoring nodes and target nodes can be verified; distances between target nodes cannot be determined. This issue can be overcome with mesh data sets. We acknowledge the fact that finding representative mesh data sets is not an easy task. However, planetary-scale testbeds such as PlanetLab [2] provides a distributed platform not only for overlay-based services but also for network measurements.

We used RTT data[1] collected between all PlanetLab nodes from March 22^{nd} to March 28^{th} 2004. During this time period, there were over 150 participating sites with a total of over 370 hosted nodes distributed in the Americas, Europe, Asia and Australia. We then calculated the minimum RTT between each pair of nodes available between those dates on consecutive 15-minute periods. Thus, for each day in this period there were 96 matrices of RTT measurements, and the size of each matrix was 325×325. Therefore, over the seven days period, we obtained 672 such matrices.

Since we wanted to perform this analysis only over our estimate of propagation delays on the paths (and not the queueing and congestion effects), we first constructed a single distance matrix D, in which each entry represented the minimum RTT between a pair of PlanetLab nodes over the entire 7-day period. This avoided biases in the results due to high variations in RTTs, e.g. during congested periods. Our analysis of the data indicated that by taking the minimum over a 7-day period, we can filter out congestion related effects which have periodic weekly patterns.

Many PlanetLab sites had multiple nodes per site. For instance, the Computer Laboratory (University of Cambridge) site hosted three nodes `planetlab1`, `planetlab2` and `planetlab3` under the domain `cl.cam.ac.uk`. The minimum RTTs between nodes within a site were very small, often of the order of 1 ms.


(a) North America (*Lipschitz* Superspace embedding). PlanetLab site with maximum rrl.

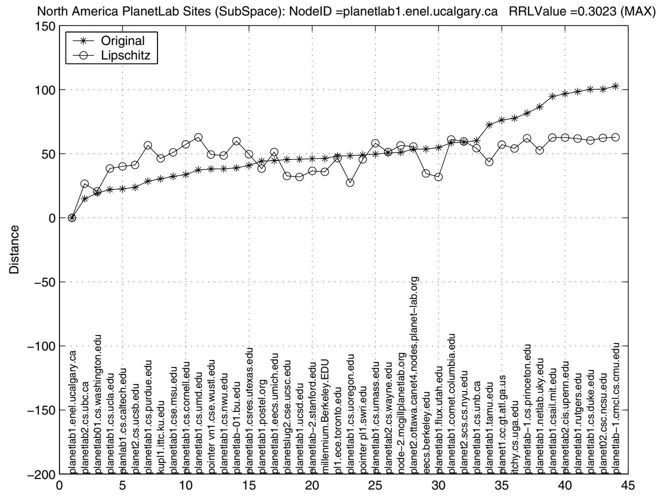

(b) North America (*Lipschitz* Subspace embedding). PlanetLab site with maximum rrl.

Figure 9: PlanetLab site with maximum rrl using full *Lipschitz* Superspace and Subspace embeddings.

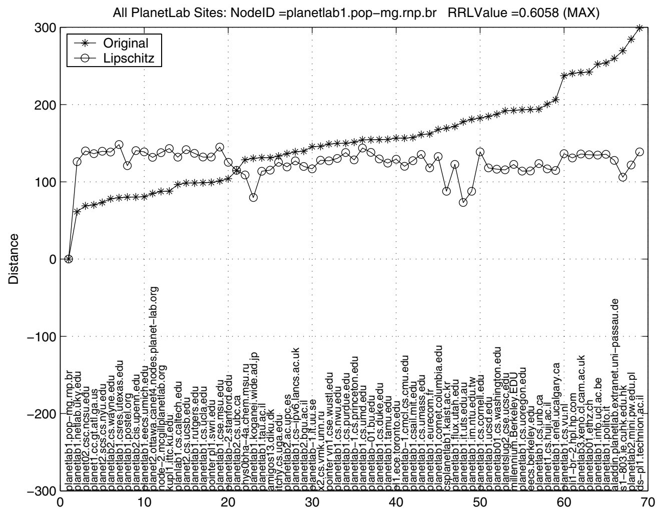


Figure 8: PlanetLab site `planetlab1-pop-mg.mp.br` with maximum rrl using *full* Lipschitz embedding.

Examining RTTs between all pairs of nodes was therefore wasteful and biased our results by the distribution of nodes in PlanetLab sites and by the fact that there were missing entries on the original 325×325 distance matrix. Thus, we selected a representative node in each site so as to build a *site by site* distance matrix D', and then we removed rows and columns that contained missing entries. This process reduced the distance matrix to 69×69 RTTs between sites. Finally, we used the classification proposed in [5] to split D' into three sub-matrices of different sizes based on the geographical locations of PlanetLab sites:

North America (NA-PL): 44×44 distance matrix which contains RTTs between PlanetLab sites located in North America. The majority of these sites obtain interconnectivity through the Abilene network.

Outside North America (ONA-PL): 25×25 distance matrix with RTTs between *research* and *commercial* sites outside North America. It includes sites in Australia, Europe, Latin America and Asia.

All sites (ALL-PL): the combination of NA-PL and ONA-PL data sets, of size 69×69.

We apply the *full* Lipschitz embedding to the RTT distance matrix of the 69 PlanetLab sites in **ALL-PL**. Here are the minimum, mean, and maximum values we found for local rrl:

	Min	$Mean$	Max
rrl	0.0887	0.1822	0.6058

The difference between the maximum and minimum rrls is high (51.71%). Figure 7 presents data from the PlanetLab site `comet.columbia.edu`, which had the minimum local rrl measure. The x-axis enumerates sites and the y-axis shows the RTT distance of each site from `comet.columbia.edu` (in milliseconds). The signature plot marked with *****'s indicates RTTs in the original distance matrix, and the sites on the x-axis have been sorted to ensure that this plot is in *ascending* order of RTTs. The signature plot marked with **o**'s is the geometric distances between the same sites in the embedded geometric space. For these signature plots, we have multiplied the Lipschitz embedding by a scaling factor that minimizes relative error (as described in Section 3). Of course, this does not impact the rrl or cnl measures, but it helps us visualize the differences between distances in the original and embedded spaces. Note that even in this best case (at least with respect to local rrl), we see that the relative ranking of many close nodes is **reversed** and the list of close neighbors is **not preserved**. We saw similar results for binary trees (Section 3), and we suspect that many of the inaccuracies visible in Figure 7 may be a consequence of the inaccuracies inherent in the Lipschitz embedding for the underlying network topology. Note that a user of this embedding at `comet.columbia.edu` would conclude that `nyu.edu` was about as far away as `utexas.edu`!

Table 1: Local rrl of *Lipschitz* Subspace and Superspace embeddings using NA-ALL and ALL-PL PlanetLab sites.

Embeddings	**Min** rrl	**Mean** rrl	**Max** rrl
ϕ(NA-PL)	0.1141	0.1897	0.3023
ϕ(NA-ALL) $\downarrow$ NA-PL	0.1606	0.2916	0.4452

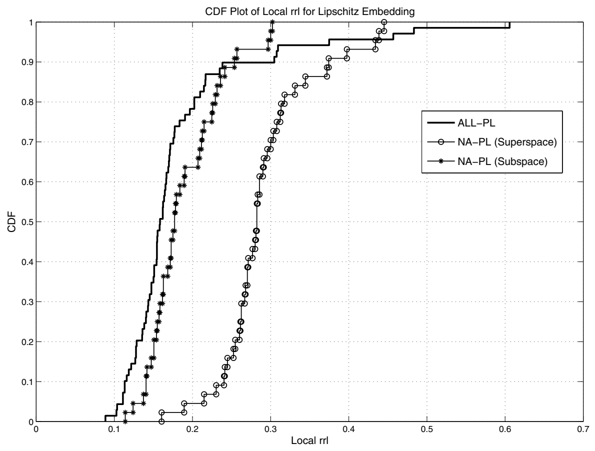


Figure 10: CDFs of Local rrl for *Lipschitz* Subspace and Superspace embeddings in Euclidean Space.

Figure 8 presents a signature plot for the site with the maximum local rrl measure, where the results here are not very impressive — site `pop-mg.rnp.br` has a 60.6% chance of swapping the relative distance of any two nodes. It would appear from looking at this plot that using this embedding at site `pop-mg.rnp.br` would give very poor results. Note that in these figures, the site has **lost** its list of neighbors' rankings in the embedding. In fact, the global cnl measure is 84.06%, for ALL-PL data set, so only about 15% of the sites retain their closest neighbors in the embedding.

We proceed to explore scalability (meta-) metric of the Lipschitz embedding with our PlanetLab site data. We used the data sets NA-PL (North America) as a Subspace of the data set ALL-PL (All sites) of size 69. We call ϕ(NA-PL) the **Subspace** embedding and ϕ(NA-ALL) $\downarrow$ NA-PL the **Superspace** embedding of NA-PL. The minimum, mean, and maximum local rrl for the full *Lipschitz* Superspace and Subspace embeddings are shown in Table 1. This is visualized in Figure 9(a) and Figure 9(b), which show the nodes with the maximum rrls for *Lipschitz* Superspace and Subspace embeddings. Note that the *Lipschitz* Subspace embedding in Euclidean space, is a **much better** one. Figure 10 shows the CDFs of the local rrl, for the *Lipschitz* ϕ(ALL-PL), ϕ(NA-PL) (Subspace), and ϕ(NA-ALL) $\downarrow$ NA-PL (Superspace) embeddings. It clearly shows that the *Lipschitz* Subspace embedding in **Euclidean space** is **better** than the *Lipschitz* Superspace embedding, at least with respect to the local rrl measure of accuracy.

5 Using Other Embeddings

We run experiments to apply our new accuracy metrics using Vivaldi and BBS (Euclidean and Hyperbolic) systems and our PlanetLab sites data. The **ALL-PL** (All sites) data set of 69 sites consisting of North America and Outside North America PlanetLab sites, are selected for this experiments. We utilize the *p2psim* simulator [1] (which Vivaldi embedding system is incorporated in), BBS (Euclidean) and BBS (Hyperbolic) TP and LRN embeddings systems, to generate 3-dimensional coordinates for our PlanetLab **ALL-PL** (All sites) data set. We utilize the TP embedding's parameters of $t = 15$ landmarks (the first 15 nodes in the data set chosen as landmarks) and 15 node-to-landmark measurements (each of the other nodes measure their distance to 15 landmarks for distortion minimization). Table 2 shows the maximum, mean and minimum rrl and cnl accuracy metrics for Vivaldi and BBS (Euclidean and Hyperbolic spaces) embeddings using **ALL-PL** PlanetLab sites data.

Table 2: rrl and cnl accuracy metrics for Vivaldi and BBS (Euclidean and Hyperbolic) embeddings using ALL-PL PlanetLab sites

Embeddings	**Min** rrl	**Mean** rrl	**Max** rrl	cnl (%)
Vivaldi	0.0817	0.1476	0.3494	75.36
BBS (Euclidean)	0.0768	0.1263	0.2291	75.36
BBS TP (Hyperbolic)	0.0382	0.2391	0.6286	72.46
BBS LRN (Hyperbolic)	0.0680	0.1419	0.3670	68.12

Both BBS (Euclidean) and Vivaldi embeddings in Euclidean space have the **same** cnl measure of 75.36%, but Vivaldi embedding has a **higher** maximum rrl compared to BBS (Euclidean). The BBS (Hyperbolic) TP embedding has a **much higher** maximum rrl than BBS (Hyperbolic) LRN embedding, but its minimum rrl is **lower** than BBS (Hyperbolic) LRN embedding. BBS (Hyperbolic) LRN embedding has the **lowest** cnl measure of 68.12%, compared to the other embeddings. On the other hand, BBS (Euclidean) embedding has the **lowest** maximum rrl and BBS (Hyperbolic) TP embedding has the **largest** maximum

rrl compared to the other embeddings. Figures 11 and 12 show the signature plots of the PlanetLab node with maximum rrl for BBS (Hyperbolic) TP embedding in Hyperbolic space and Vivaldi embedding in Euclidean space respectively. Both graphs show their lists of close neighbors are being **pushed away** in the embedded target geometric space.

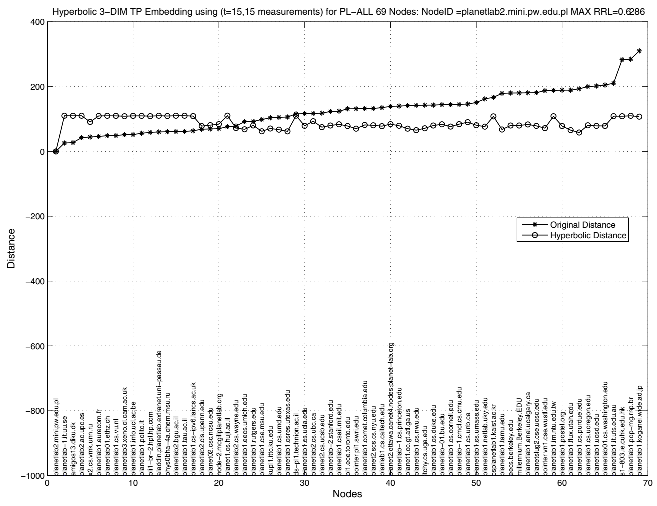


Figure 11: BBS (Hyperbolic) TP embedding using 69 PlanetLab **ALL-PL** (All sites) network topology — Nodes with Maximum rrl.

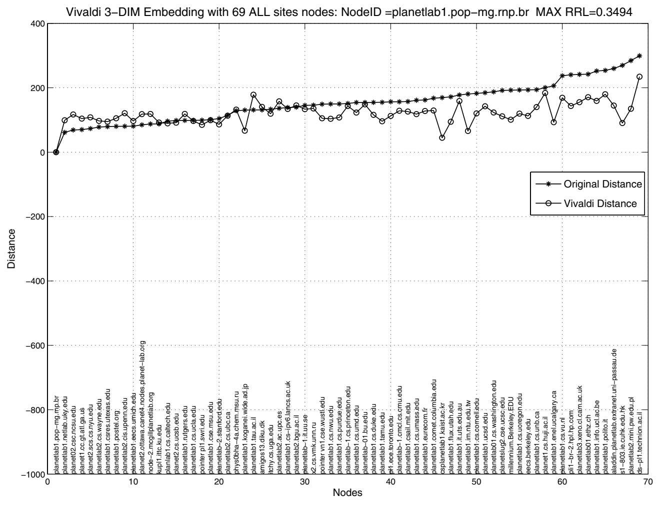


Figure 12: Vivaldi embedding using 69 PlanetLab **ALL-PL** (All sites) network topology — Nodes with Maximum rrl.

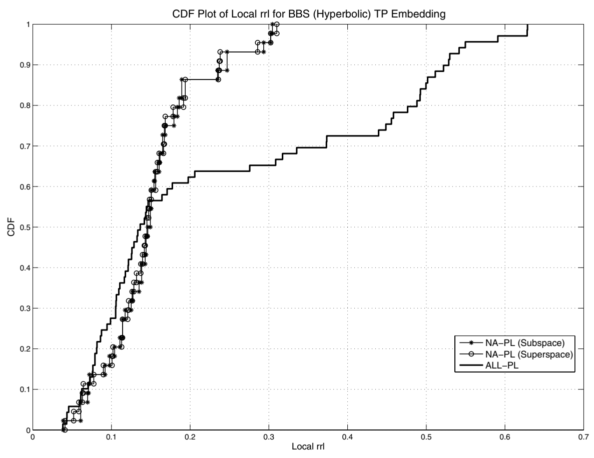
Similarly, we explore scalability (meta-) metric of the Vivaldi and BBS embeddings with our PlanetLab site data. The *Vivaldi* and *BBS (Euclidean)* Subspace and Superspace embeddings have the same behavior as the *Lipschitz* Subspace and Superspace embeddings, i.e. the Subspace embedding has **better** rrl accuracy than the Superspace embedding in the **Euclidean space**. However, in *BBS (Hyperbolic) TP* and *BBS (Hyperbolic) LRN* Subspace and Superspace embeddings, the Subspace embedding rrl accuracy is **close** to the Superspace embedding as shown in the Table 3. For BBS (Hyperbolic) TP embedding using the NA-PL site data set, we adopt the same parameters as in the TP embedding for the ALL-PL, which is $t = 15$ landmarks (the first 15 nodes in the data set chosen as the landmarks), and 15 node-to-landmark measurements (each of the other nodes measure their distance to 15 landmarks for distortion minimization). Figure 13 shows the CDFs of the local rrl for the *BBS (Hyperbolic) TP* and *LRN* ϕ(ALL-PL), ϕ(NA-PL) (Subspace) and ϕ(NA-ALL) $\downarrow$ NA-PL (Superspace) embeddings. This results suggest that the Superspace embedding tends to have a **close** or **better** rrl accuracy than the Subspace embedding in the **Hyperbolic space**.

Table 3: Local rrl for *Vivaldi* and *BBS* Subspace and Superspace embeddings using NA-ALL and ALL-PL PlanetLab sites

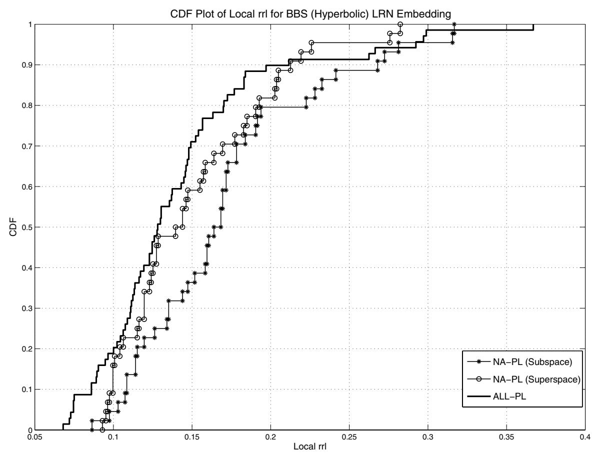
Embeddings	**Min** rrl	**Mean** rrl	**Max** rrl
Vivaldi			
ϕ(NA-PL)	0.0853	0.1542	0.2625
ϕ(NA-ALL) $\downarrow$ NA-PL	0.1351	0.23	0.4507
BBS (Euclidean)			
ϕ(NA-PL)	0.0819	0.1407	0.2614
ϕ(NA-ALL) $\downarrow$ NA-PL	0.1440	0.2170	0.3544
BBS (Hyperbolic) TP			
ϕ(NA-PL)	0.0388	0.1512	0.3045
ϕ(NA-ALL) $\downarrow$ NA-PL	0.0410	0.1496	0.3101
BBS (Hyperbolic) LRN			
ϕ(NA-PL)	0.0864	0.1713	0.3167
ϕ(NA-ALL) $\downarrow$ NA-PL	0.0930	0.1504	0.2824

6 Revisiting Previous Work with New Accuracy Metrics

It will be interesting to apply our new accuracy metrics to Vivaldi and BBS (Euclidean and Hyperbolic) systems (based on numerical minimization techniques) on *their* data sets that were used in their work. We generate 3-dimensional coordinates for these embeddings in our experiments. With reference to BBS systems [18–20], we utilize network topologies from the TRACER placement method (the network topologies chosen by selecting the first n nodes in the *whole* network) for our experiments due to its suitability for geometric distance experiments. We utilize the TP embedding's parameters of $t = 10$ landmarks and 6 node-to-landmark measurements: this means that there are 10 landmarks chosen from the first 10 nodes in the network topology and *only* 6 *closest* measurements are carried out by other nodes to these 10 landmarks.



(a) CDFs of Local rrl for *BBS (Hyperbolic) TP* Subspace and Superspace embeddings.



(b) CDFs of Local rrl for *BBS (Hyperbolic) LRN* Subspace and Superspace embeddings.

Figure 13: CDFs of Local rrl metrics for Subspace and Superspace embeddings in Hyperbolic Space.

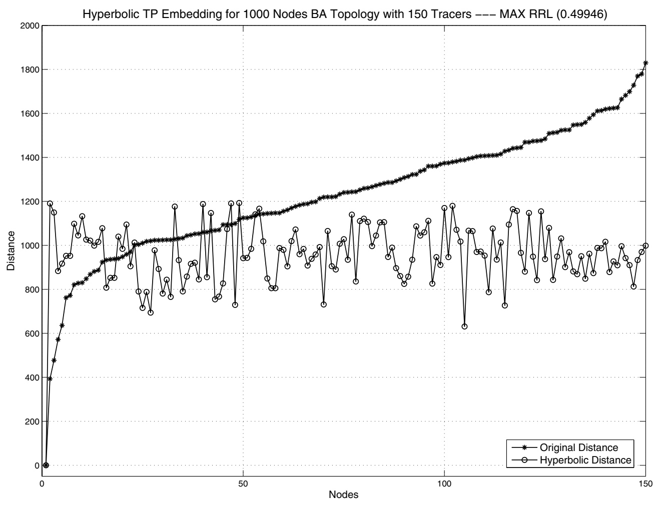
Table 4: rrl accuracy metric for Vivaldi and BBS (Euclidean and Hyperbolic) embeddings

Embeddings	**Data sets**	**Min** rrl	**Mean** rrl	**Max** rrl
Vivaldi	192 PlanetLab filtered nodes	0.0334	0.0894	0.2808
Vivaldi	1740 King nodes	0.0633	0.1306	0.5667
BBS (Euclidean)	Topology of 50 nodes chosen among 600 nodes in Waxman Network	0.1310	0.2090	0.3639
BBS (Euclidean)	Topology of 15 nodes chosen among 1000 nodes in BA Network	0.0220	0.0982	0.2857
BBS (Euclidean)	Topology of 150 nodes chosen among 6474 nodes in Jan 2000 AS Network	0.3531	0.4553	0.6379
BBS (Hyperbolic) TP with 10 landmarks and 6 measurements	Topology of 150 nodes chosen among 1000 nodes in BA Network	0.0892	0.2452	0.4995
BBS (Hyperbolic) TP with 10 landmarks and 6 measurements	Topology of 150 nodes chosen among 6474 nodes in Jan 2000 AS Network	0.3305	0.4636	0.6452
BBS (Hyperbolic) TP with 10 landmarks and 6 measurements	Topology of 50 nodes chosen among 600 nodes in Waxman Network	0.1573	0.2647	0.4524
BBS (Hyperbolic) TP with 15 landmarks and 15 measurements	Topology of 200 nodes with *lower* degree chosen among 10670 nodes in Mar 2001 AS Network	0.0228	0.1419	0.2153
BBS (Hyperbolic) LRN	Topology of 200 nodes with *lower* degree chosen among 10670 nodes in Mar 2001 AS Network	0.0802	0.1796	0.2389

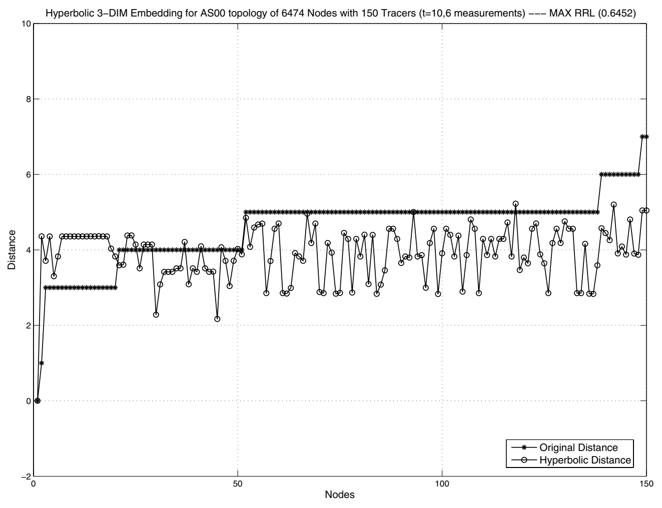
There are *two* data sets used in the Vivaldi [8] work. **PlanetLab RTT data of filtered nodes**: This is pair-wise minimum RTTs of 192 PlanetLab nodes over a certain period of time (not specified in [8]), from the PlanetLab ”ping” collection traces. It is filtered due to some nodes in PlanetLab having security measures restricting ”ping” connection from other nodes. **King [9] data**: This data set contains 1740 Internet DNS servers RTT traces based on the King collection method.

We selected the following data sets from the BBS (Euclidean) [18] work. **Waxman [22] Network Topology**: This data set contains undirected fixed network topologies of 600 nodes and the edge weights are generated with random uniform distribution in the range of $[1, 1000]$. The edge weights are taken as $\lfloor 10^E + 0.5 \rfloor$, where E is randomly and uniformly distributed in the interval $(0, 3]$. Based on the 600 nodes' connectivity, a Dijkstra algorithm is performed to find the shortest path distance matrix. Network topologies consist of the first 50 nodes chosen among the 600 nodes in the Waxman Network. **Barabási-Albert [4, 6] (BA) Network Topology (considered to follow the PowerLaw)**: This data set consists of undirected fixed network topologies of 1000 network nodes and the edge weights are generated with random uniform distribution in the interval of $[1, 1000]$. Based on the 1000 nodes' connectivity, a Dijkstra algorithm is performed to find the shortest path distance matrix. Network topologies consist of the first 15 nodes chosen among the 1000 nodes in the BA Network.

We used the following data sets from BBS (Hyperbolic) [19, 20] work. **Barabási-Albert (BA) [4, 6] Network Topology (considered to follow the PowerLaw)**: This data set consists of undirected fixed network topologies of 1000 nodes, and edge weights are generated with random uniform distribution in the interval of $[1, 1000]$ (same data set in BBS (Hyperbolic) work [19]). Network topologies



(a) Node with Maximum rrl using network topology of 150 nodes chosen from 1000 nodes in BA Network Topology.



(b) Node with Maximum rrl in network topology of 150 nodes chosen from 6474 nodes in Jan 2000 AS Hierarchical Tree Network Topology.

Figure 14: BBS (Hyperbolic) TP embedding with 10 landmarks and 6 measurements — Nodes with Maximum rrl.

consist of the first 150 nodes chosen among the 1000 nodes. A Dijkstra algorithm is performed to find the shortest path distance matrix. **January** 2000 **AS Network Topology from University of Oregon (RouteViews)**: This hierarchical tree network topology consists of 6474 nodes, and edge weights are generated with random uniform distribution in the interval of $[1, 1000]$ (same data set in BBS (Hyperbolic) work [19]). Network topologies consist of the first 150 nodes chosen among the 6474 nodes. A Dijkstra algorithm is performed to find the shortest path distance matrix. **March** 2001 **AS Network Topology from University of Oregon (RouteViews)**: This data set is only used in the BBS (Hyperbolic) LRN embedding work [20]. It consists of a hierarchical tree network topology of 10670 nodes that are randomly weighted, with edge weights distributed uniformly in the interval of $[1, 1000]$. We extracted 200 weighted network topology of nodes with lower degree, or stub ASes.

Table 4 summarizes the maximum, mean and minimum rrl accuracy metric for Vivaldi and BBS (Euclidean and Hyperbolic) embeddings using their data sets [8, 18–20]. The signature plots of the nodes having maximum rrl for BBS (Hyperbolic) TP embedding in Hyperbolic space are shown in Figure 14. Evidently, the result of TP embedding in Hyperbolic space using AS Hierarchical Tree Network Topology shows **similar** rrl inaccuracy behavior as the Lipschitz embedding in Euclidean space from the perspective of the tree node in the synthetic binary trees which we have illustrated in Section 3.5. This confirms that BBS (Hyperbolic) TP embedding in **Hyperbolic space** has the **similar** rrl inaccuracy behavior as *Lipschitz* embedding in **Euclidean space** for **tree-like** network topology. Also, from the BBS (Hyperbolic) TP embedding experiment using BA Network Topology, it shows **similar** rrl inaccuracy behavior as the Lipschitz embedding using our PlanetLab site data. All these experiments show that the lists of close neighbors are being **pushed away** in the target Euclidean and Hyperbolic spaces.

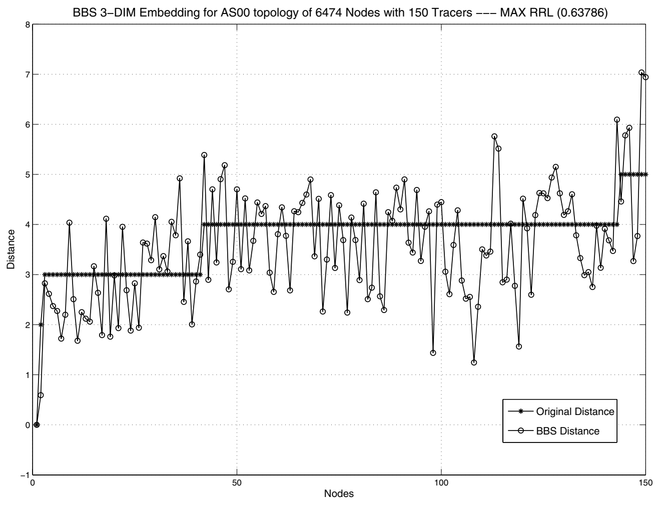
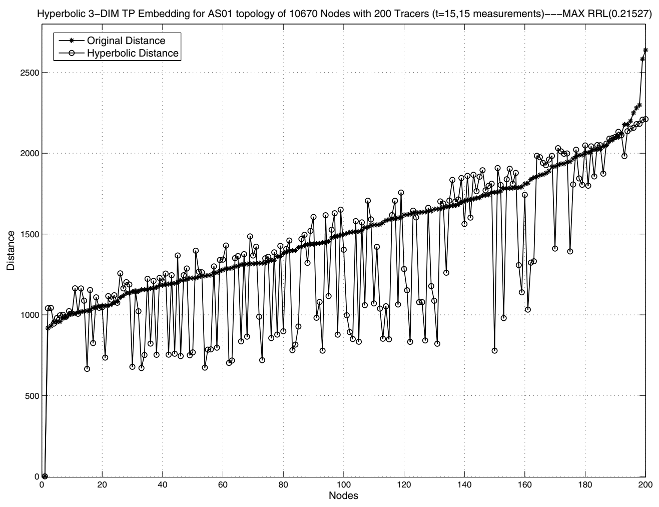
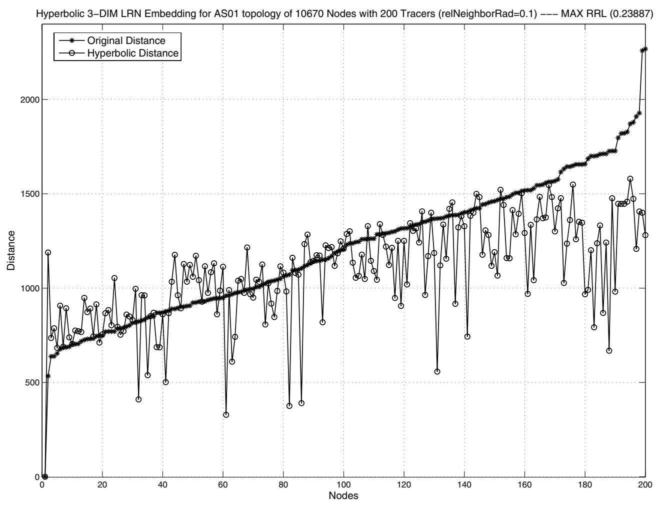


Figure 15: BBS (Euclidean) embedding using network topology of 150 nodes chosen from 6474 nodes in Jan 2000 AS Hierarchical Tree Network Topology — Node with Maximum rrl.

We attempt to make a comparison study of BBS (Euclidean) and BBS (Hyperbolic) TP embeddings. The Jan 2000 AS Hierarchical Tree Network Topology (RouteViews) data set is used for this comparison. We examine



(a) Node with Maximum rrl for BBS (Hyperbolic) TP embedding.



(b) Node with Maximum rrl for BBS (Hyperbolic) LRN embedding.

Figure 16: rrl Comparison of BBS (Hyperbolic) TP embedding with 15 landmarks and 15 measurements and BBS (Hyperbolic) LRN embedding, using network topology of 200 nodes with lower degree chosen from 10670 nodes in Mar 2001 AS Hierarchical Tree Network Topology.

the signature plot of nodes with the maximum rrl for BBS (Hyperbolic) TP and BBS (Euclidean) embeddings, as illustrated in Figure 14(b) and Figure 15 respectively. For hierarchical tree-like network graphs, both embeddings in Euclidean and Hyperbolic spaces have **sharp bi-modal** rrl error behaviors. Similarly, in **Euclidean space**, the rrl measure for the BBS (Euclidean) embedding shows **similar** rrl inaccuracy behavior as the Lipschitz embedding from the perspective of the tree node in the synthetic binary trees, which we have illustrated in Section 3.5.

We continue to run rrl accuracy metric on BBS (Hyperbolic) LRN embedding using the March 2001 AS Hierarchical Tree Network Topology (RouteViews) data set. This is compared with BBS (Hyperbolic) TP embedding's parameters of $t = 15$ landmarks, and 15 node-to-landmark measurements for distortion minimization. Figure 16 shows the signature plots of the nodes with maximum rrl for BBS (Hyperbolic) TP and LRN embeddings. In LRN embedding, the list of close neighbors is being **pushed away very much further** and it has a **higher** maximum rrl compared to TP embedding. Both exhibit **similar bipolar** rrl error behavior in Hyperbolic space.

7 Discussion

Our goal in this experimental work is to apply our new accuracy metrics to study the *accuracy* of estimated distances predicted by Internet coordinate systems and to what extent it varies among different pairs of nodes. The results of this initial attempt are not very encouraging. However, it remains that no single accuracy metric can capture the *quality* of the embedding techniques and it is worthwhile to develop a collection of accuracy metrics that are able to quantify user-oriented *quality*. Another interesting open problem is: Can we characterize the impact of network topologies that have good embeddings with respect to an accuracy metric? We envisage to answer this through our newfound future research on EONs and to discover its embeddability which aim to give good *quality* embeddings with respect to accuracy metrics. We also note that the number of landmarks has some impact on the accuracy of the embedding techniques: less information can give *better* accuracy results. Figures 17 and 18 show the accuracy metrics (average absolute relative error and rrl) versus number of land-

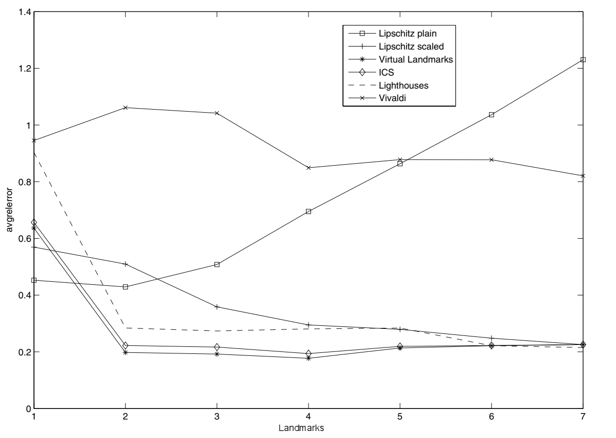


Figure 17: Average Absolute Relative Error Metric versus Number of Landmarks.

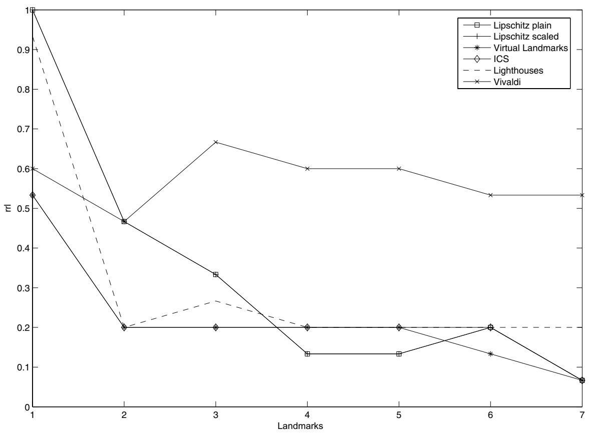


Figure 18: rrl Error Metric versus Number of Landmarks.

marks of the embedding techniques for the binary tree of depth 2 (Figure 1). The landmarks are selected sequentially and incrementally from the list of nodes in the binary tree. The results of this simple experiment indicate that Lipschitz plain embedding (without any scaling) gives *better* average relative absolute error and Vivaldi embedding gives *lower* rrl with small number of landmarks. Other embedding techniques have similar observation. This contradicts the intuition that having more landmarks (i.e. having more information) will give *better* accuracy results. A complete understanding will require the investigation of structural effects of the number of landmarks and landmarks' selection with respect to accuracy metrics.

8 Acknowledgments

Initial work for this paper was done while the first four authors were with Intel Research, and we thank Derek McAuley for his support. We also thank Yuval Shavitt, Tomer Tankel, Jinyang Li and Frank Dabek for useful and helpful discussions of their systems and data sets. Eng Keong Lua is sponsored by Microsoft Research Ph.D. Studentship.

References

[1] p2psim: A simulator for peer-to-peer protocols. http://www.pdos.lcs.mit.edu/p2psim/index.html.

[2] PlanetLab home page. http://www.planetlab.org.

[3] Skitter home page. http://www.caida.org/tools/measurement/skitter.

[4] ALBERT, R., AND BARABÁSI, A.-L. Topology of evolving networks: Local events and universality. *Physical Review Letters* (December 11 2000), 5234–5237.

[5] BANERJEE, S., GRIFFIN, T. G., AND PIAS, M. The interdomain connectivity of planetlab nodes. In *Proceedings of the 5th International Workshop on Passive and Active Network Measurement* (Antibes Juan-les-Pins, France, April 2004).

[6] BARABÁSI, A.-L., AND ALBERT, R. Emergence of scaling in random networks. *Science 286* (October 15 1999), 509–512.

[7] COSTA, M., CASTRO, M., ROWSTRON, A., AND KEY, P. Pic: Practical internet coordinates for distance estimation. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS' 04)* (Tokyo, Japan, March 2004).

[8] DABEK, F., COX, R., KAASHOEK, F., AND MORRIS, R. Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM 2004* (Portland, Oregon, August 2004).

[9] GUMMADI, K., SAROIU, S., AND GRIBBLE, S. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the Internet Measurement Workshop* (November 2002).

[10] GUPTA, A. Embedding tree metrics into low dimensional euclidean spaces. In *Proceedings of the 31st annual ACM symposium on Theory of computing* (1999), pp. 694–700.

[11] LIM, H., HOU, J., AND CHOI, C. Constructing internet coordinate system based on delay measurement. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC 2003)* (Miami, Florida, USA, October 2003).

[12] LINIAL, N., LONDON, E., AND RABINOVICH, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica 15* (1995), 215–245.

[13] LINIAL, N., MAGEN, A., AND SAKS, M. Trees and euclidean metrics. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)* (1998).

[14] LINIAL, N., AND SAKS, M. The euclidean distortion of complete binary trees. *Discrete and Computational Geometry 29*, 1 (2003), 19–21.

[15] MAO, Y., AND SAUL, L. K. Modeling distances in large-scale networks by matrix factorization. In *Proceedings of the 4th ACM SIGCOMM conference on Internet Measurement* (2004), pp. 278–287.

[16] NG, T. E., AND ZHANG, H. Predicting internet network distance with coordinates-based approaches. In *Proceedings of the IEEE INFOCOM 2002* (New York, USA, June 2002).

[17] PIAS, M., CROWCROFT, J., WILBUR, S., HARRIS, T., AND BHATTI, S. Lighthouses for scalable distributed location. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems* (February 2003).

[18] SHAVITT, Y., AND TANKEL, T. Big-bang simulation for embedding network distances in euclidean space. In *Proceedings of the IEEE INFOCOM 2003* (San Francisco, California, April 2003).

[19] SHAVITT, Y., AND TANKEL, T. On the curvature of the internet and its usage for overlay construction and distance estimation. In *Proceedings of the IEEE INFOCOM 2004* (Hong Kong, March 2004).

[20] SHAVITT, Y., AND TANKEL, T. On internet embedding in hyperbolic spaces for overlay construction and distance estimation. *Submitted to a Journal,* `http://www.eng.tau.ac.il/~tankel/pub/HypEmb05v2.pdf` (2005).

[21] TANG, L., AND CROVELLA, M. Virtual landmarks for the internet. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC 2003)* (Miami, Florida, USA, October 2003).

[22] WAXMAN, B. Routing of multipoint connections. *IEEE JSAC* (1988), 1617–1622.

[23] ZHENG, H., LUA, E. K., PIAS, M., AND GRIFFIN, T. Internet routing policies and round-trip-times. In *Proceedings of the Passive Active Measurement 2005* (March 30 - April 1 2005).

Notes

[1]The all-pairs RTT measurement data was being continuously collected by Jeremy Stribling and it is available from `http://www.pdos.lcs.mit.edu/~strib/pl_app/`

An Empirical Approach to Modeling Inter-AS Traffic Matrices

Hyunseok Chang
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
hschang@eecs.umich.edu

Sugih Jamin*
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
jamin@eecs.umich.edu

Z. Morley Mao†
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
zmao@eecs.umich.edu

Walter Willinger
AT&T Labs-Research
180 Park Ave.
Florham Park, NJ 07932-0971
walter@research.att.com

Abstract

Recently developed techniques have been very successful in accurately estimating *intra-Autonomous System (AS)* traffic matrices. These techniques rely on link measurements, flow measurements, or routing-related data to infer traffic demand between every pair of ingress-egress points of an AS. They also illustrate an inherent mismatch between data needed (e.g., ingress-egress demand) and data most readily available (e.g., link measurements). This mismatch is exacerbated when we try to estimate *inter-AS* traffic matrices, i.e., snapshots of Internet-wide traffic behavior over coarse time scale (a week or longer) between ASs. We present a method for modeling inter-AS traffic demand that relies exclusively on publicly available/obtainable measurements. We first perform extensive Internet-wide measurement experiments to infer the "business rationale" of individual ASs. We then use these business profiles to characterize individual ASs, classifying them by their "utility" into ASs providing Web hosting, residential access, and business access. We rank ASs by their utilities which drive our gravity-model based approach for generating inter-AS traffic demand. In a first attempt to validate our methodology, we test our inter-AS traffic generation method on an inferred Internet AS graph and present some preliminary findings about the resulting inter-AS traffic matrices.

1 Introduction

Motivated by recent successes of *intra-domain* traffic matrix estimation techniques [8, 23, 16], we attempt to obtain accurate estimates of traffic volume exchanged between individual ASs. Knowledge of such a global but coarse-scale spatio-temporal picture of Internet traffic is vital for a number of practical networking problems, including evaluating the impact of emerging technologies such as "intelligent routing control" (e.g., multi-homing, overlay routing) [3]; identifying or forecasting potential network bottlenecks or investigating the effectiveness of proposed remedies aimed at alleviating growing congestion [2]; assessing the performance of new protocols or Internet-wide applications such as new overlay networks; or improving the current understanding of how intra-domain traffic engineering (TE) impacts inter-domain TE and vice versa [1, 15, 14]. In the spirit of the work by Feldman et al. [9], we can use these inter-domain traffic matrices to partially answer such questions as "How stable are AS traffic volumes over time?" or "Which ASs carry most of the traffic, and how much?"

Given the highly competitive nature of today's Internet Service Provider (ISP) market, ISPs do not make public such sensitive data as traffic volume statistics. The task of capturing the global behavior of AS-level Internet over long time scale (e.g., days) has been left to experimentalists. Due to the immense difficulties in collecting high volume of traffic data on an Internet-wide scale, research efforts to model and estimate inter-domain traffic demand are still in their infancy. With some exceptions [6, 22, 9], most studies that require knowledge of inter-domain traffic demand typically employ extremely simple (and untested) demand models, often assuming uniform traffic demand between every pair of ASs [2, 21]. Studies that rely on traces collected from a single vantage point (typically located in some stub network) are inherently constrained in their ability to provide a global view of inter-domain traffic. Nevertheless, an analysis of these traces revealed that while any given AS may exchange traffic with most of the Internet, only a small number of ASs are responsible for a large fraction of inter-domain traffic. In contrast to [6, 22], Feldmann et al. [9] also use server logs from a large CDN and develop a methodology for estimating inter-domain demand matrices for Web traffic.

Our model of inter-domain traffic demand encompasses the overall traffic, with Web traffic representing just one (though quite significant) component. Considering the

*This project is funded in part by NSF grant number ANI-0082287 and by ONR grant number N000140110617. Sugih Jamin is further supported by the NSF CAREER Award ANI-9734145, and the Presidential Early Career Award for Scientists and Engineers 1998.

†Z. Morley Mao is supported by the NSF Award CNS-0430204.

highly restrictive nature of access to proprietary data (such as server logs of large CDNs), we rely only on publicly available or obtainable data in our modeling of inter-domain traffic demand. However, the proposed approach is flexible enough to incorporate proprietary data, should it become available and should it be relevant to the problem at hand. In addition, we argue that a useful inter-domain traffic demand model should possess the following two characteristics. First, given the challenging task of obtaining actual inter-domain traffic matrices, the model should be *flexible* enough to allow for a systematic exploration of different traffic scenarios and traffic engineering strategies. At the same time, the model should be *parsimonious* enough to provide an intuitive understanding of the generated traffic demand and allow for a network-grounded interpretation of its parameters.

Our approach to develop such a model is partly empirical and partly analytical. We combine information gained from performing extensive Internet-wide measurement experiments with a modeling framework known as the "general gravity model." The general gravity model has recently been used in estimating *intra*-domain traffic matrices [23] and has a long history in the social sciences, where it has been applied to describe the movement of people, goods, and information between geographic regions [17, 7]. To apply the gravity model to inter-domain traffic modeling, we first need to define the concepts of "mass" and "distance" within the context of inter-domain traffic exchange. To that end, we start by discussing the *business model (or operational characteristics)* of individual ASs.

In Section 2 we make the null hypothesis that to a first approximation, the traffic volume exchanged between two ASs necessarily reflects the business model of their operators. For example, an AS in the business of hosting various web and multimedia content will exhibit a very lopsided traffic profile (i.e., disproportionately heavy outbound traffic volumes). For another example, if two ASs are mainly in the business of providing access to residential customers, with comparable customer bases, traffic demand between the two networks can be expected to be more symmetric. By exploiting a range of publicly available data sets and by relying on information collected from our own Internet-wide experiments, we develop in Section 3 a combined measurement and business "profiling" methodology to infer the "utility" of an AS's physical network. We identify the utility of an AS as providing Web hosting, residential access, or business access services. Depending on a simple high/low classification of these utility values, we infer seven natural AS business models. In Section 3, we classify ASs into one of these models, and rank the ASs within each class by their combined utility. These rankings then constitute the key input data to our general gravity model presented in Section 5 and determine the generation of inter-domain traffic volumes exchanged between individual ASs. In particular, we illustrate that our model, besides being flexible and parsimonious, is capable of generating realistic traffic demand with different characteristics. As a final contribution, we focus in Section 6 on model validation and attempt to partially address this issue, even though inter-domain traffic matrix estimation is a case where even the most basic "ground truth" appears elusive. We conclude in Section 7 with a discussion of unresolved issues and open problems.

2 AS Business Models

We broadly define an AS's business model as the *utility* of its physical networks, i.e., the primary reason(s) behind the design, operation, and management of its physical infrastructure. We associate a "business model" with each AS, not with each ISP or company, mainly because some large ISPs maintain multiple ASs (or domains) and typically assign these (sub)domains to separate business divisions, each with its own business characteristics or utility. We avoid the daunting task of identifying and enumerating the business purposes and operating strategies of all ASs by restricting our attention to the most *generic* utilities of existing networks that clearly affect their resulting traffic demand. In the following, we identify three such utilities (i.e., Web hosting, residential access, and business access) and describe how they can impact inter-domain traffic demand.

Web Hosting (Web). The success of the World Wide Web has led to an explosion of web sites that host various web content and streaming media. Powered by sophisticated content distribution technologies, a number of Web hosting companies have also emerged to host content outsourced by popular web sites. So much so that Web hosting is now a common service included in an AS's service portfolio. An AS that hosts popular web content or e-commerce engines and distributes this content to the global Internet can be expected to carry voluminous outbound traffic and relatively little inbound traffic.

Residential Access (RA). Retail Internet business that directly deals with residential customers has existed since the inception of the commercial Internet. With advances in Internet access technologies, along with access speed, the number of residential users equipped with Internet access has risen steadily.[1]With the proliferation of high-speed Internet users, the influence of end-user applications on the global traffic pattern becomes increasingly pronounced. A prime example are the bandwidth-demanding peer-to-peer (P2P) file-sharing applications. ASs populated by a large pool of residential users can exchange nontrivial amount of traffic among themselves as well as receive a large amount of web download traffic from Web content networks.

Business Access (BA). The low barrier of entry into the ISP market creates an environment whereby wholesaling and reselling of Internet access is actively pursued by both

Table 1: AS business profiles and models

Utility Profile			AS Business Model
Web	RA	BA	
H	H	H	Tier-1
H	H	L	Retail service
H	L	H	Business service
L	H	H	Network access
H	L	L	Web hosting
L	H	L	Residential access
L	L	H	Business access

incumbent carriers and new market entrants. Customers of large ISPs with nation- or continent-wide footprints are often themselves ISPs that resell the purchased Internet access to their own customers. For the purposes of this paper, "business access providers" are ISPs that resell purchased Internet access. Traffic demand of ASs that are business access providers can be estimated by the quality of service these ASs provide. Transit ASs guaranteeing good quality of service are likely to attract and retain a high number of customers. In turn, customers with reliable Internet connections can rely on the Internet for a large part of their business transactions, resulting in high traffic demand.

While the first two of these three utilities are traditionally attributed to stub networks, the third one is typically associated with transit networks. In today's ISP market the service portfolio of large ISPs typically reflect multiple concurrent utilities (e.g., a business access provider may also be in the business of providing residential access or Web hosting services). We therefore do not follow the traditional classification of ASs into stub and transit networks. Instead, we attempt to determine a given AS's business model by inferring from relevant data which of the three utilities dominate the AS's operation, and what combination of utilities best characterizes the AS's business. Table 1 lists seven AS business profiles based on the three identified utilities. Whether a given utility is primary or secondary to an AS's business profile is denoted by "H" (high) and "L" (low). We tag each of the seven business profiles with an appropriately named business model listed in the right column.

3 Method for Inferring AS Business Model

In the highly competitive ISP market, the business plans of existing ISPs are generally confidential and cannot be inspected. Commercial ISP market research studies, e.g., PriMetrica (`telegeography.com`), are often dated, cover only a handful of well-known ISPs, or are based on some very narrowly-defined criteria. They are in general ill-suited for inferring business profiles of ASs, as defined in Table 1, in a comprehensive and coherent manner. As a viable alternative, we propose a methodology for inferring an AS's business model that involves performing extensive Internet-wide measurement experiments and also involves collecting data indicative of individual ASs' utilities. We rely exclusively on publicly available/obtainable data and assume that we have no access to any proprietary data. We discuss the limitations imposed by this restriction and comment on how they could be alleviated if different types of proprietary data (e.g., server logs from a large CDN as used by Feldman et al. [9]) should become available.

Table 2: Computation of $U_{web}(\cdot)$

```
initialize $U_{web}(X)$ to 0 for every AS $X$.
For each URL $u$,
  let $size(u)$ be the size of a file referred to by $u$.
  extract a web server name $N$ from $u$.
  find the IP address set $\mathcal{S}$ that is resolved to $N$.
  For each IP address $I \in \mathcal{S}$,
    find AS $X$ that $I$ belongs to.
    $U_{web}(X) = U_{web}(X) + \frac{size(u)}{|\mathcal{S}|}$
```

3.1 Web Hosting

Our approach to quantifying web service utility is based on locating popular content on the Internet. ASs that host a large amount of popular content are considered to have high utility as a web service provider. To determine popular web content, we first consulted the web site Aleksika (`www.skyart.org`) and obtained a list of the top 10,000 search keywords most frequently submitted to search engines in the years 2003-2004. For each keyword, we queried the Google search engine, using the Google Web Services Application Programmer Interface (API), to retrieve a set of most closely matched URLs. For each submitted query, the Google Web API returned the top-10 matched results (URLs). We collected about 85,000 distinct URLs from all Google responses. By extracting web server addresses from these URLs, we inferred the ASs hosting widely accessed web content. To prevent bias towards discovering English-only content, we repeated the above experiment six more times, directing the Google Web Services API to return URLs in Chinese, French, German, Japanese, Korean, and Spanish respectively. The same set of English keywords was used in each experiment. For each language, we obtained between 80,000 and 90,000 URLs for these keywords. Merging all these results yielded close to 650,000 distinct URLs.

One source of web traffic not captured by the above measurement experiments is *embedded* web content, which includes media files delivered by dedicated media servers secondary to a web server, private CDNs, or third-party online advertisement objects (e.g., `doubleclick.com`). To uncover such web traffic, we crawled the above keyword-retrieved URLs individually, and extracted from the crawled pages URLs associated with embedded objects. Combining keyword-retrieved URLs with embedded-object URLs increased the total number of our collected URLs by a factor of three.

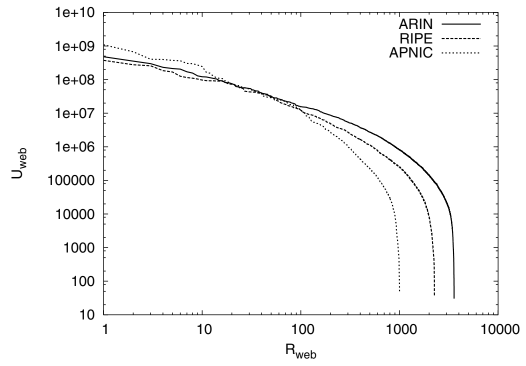


Figure 1: U_{web} distribution

Next we extracted a web server IP address from each URL and mapped it to its corresponding AS. A number of issues complicated this seemingly straightforward step. For one, some frequently-accessed web sites employ DNS-based load balancing, whereby their domain names are resolved to multiple addresses in a round-robin manner. In more sophisticated cases, a given domain name is resolved to a set of addresses depending on the querying client's geographic location, web server availability, and network condition. CNN and AOL are examples of two content providers employing such DNS-based web request routing, and their domain names are typically resolved to hundreds of addresses spread out geographically and administratively over the Internet. To obtain all IP addresses associated with a given web server's domain name, we performed a reverse-DNS lookup of each domain name from 96 geographically dispersed PlanetLab nodes (`planet-lab.org`), and collected all resulting addresses. Using BGP routing tables, we then mapped each resulting IP address to its corresponding AS. In Table 2, we summarize the steps taken to obtain the web-hosting utility $U_{web}(X)$ for every AS X, where $U_{web}(X)$ can be viewed as an estimate of the byte counts of popular web content hosted by AS X.

Finally, we sort the ASs by their utility $U_{web}(\cdot)$ in decreasing order and assign them ranks, denoted $R_{web}(\cdot)$. Table 3 lists the top-10 Web hosting ASs by rank, in three different geographic regions: North America (ARIN), Europe (RIPE) and Asia-Pacific (APNIC). As expected, the top-ranking Web hosting ASs include e-commerce companies (e.g., Amazon and eBay), telecom companies (e.g., AT&T, Deutsche Telekom, Korea Telecom), and well-known portal sites (e.g., Yahoo). One notable observation is the dominance of telecom companies in the Web hosting business in the Asia-Pacific region.

Fig. 1 shows the entire R_{web} vs. U_{web} distribution. Consistent with previous findings [6, 22, 9], $U_{web}(\cdot)$ associated with high-ranking ASs (e.g., up to rank $100 - 400$) in all three geographic regions are characterized by a Zipf-type law (i.e., $U_{web} \sim (R_{web})^c$, where $c \approx -0.9$ for ARIN and RIPE, and $c \approx -1.1$ for APNIC). The steep fall-off of the curves in the region of low-ranking ASs is likely due to the limited coverage our keyword-based crawling process has of their web content.

The key underlying assumption in our empirical method is that to infer the popularity of content networks, a feasible alternative to using actual web traffic measurement is to rely on data that measures the appearance of web content in search results. Strictly speaking, this alternative method can only account for traffic from users actively seeking specific information, not traffic from users visiting bookmarked pages or links on web pages. However, due to our decision to use the top 10,000 most popular keywords, we expect the resulting bias to be small. It is well-known that Google's PageRank weighting algorithm carefully considers web link structures (e.g., links that a page receives) in calculating link values. As a result, we believe that our method does implicitly account for some aspects of actual link traffic.

Ideally, characterizing web service utility should make use of measured inter-domain web traffic, as is done for example by Feldman et al. [9]. However, this requires access to server logs of widely-deployed private CDNs. Such data is not publicly available. These data sets are also limited in their coverage: they capture only web content served by the CDN, and they do not capture web traffic emanating from content providers who are not clients of the CDNs. These difficulties illustrate the technical challenges associated with accurate estimation of inter-domain web traffic. Viewing our method as one of many viable approaches to making progress in this area, it exemplifies how a combination of publicly obtainable and publicly available data sets can be used to infer inter-domain traffic volume. At the same time, the method is flexible enough to incorporate web server logs from CDNs should such logs become publicly available.

3.2 Residential Access

To infer an AS's utility in providing residential Internet access, we estimate it by the number of P2P file sharing users of the AS. Besides web browsing, P2P file-sharing is currently one of the most popular applications on the Internet (`cachelogic.com`). To estimate the number of users per AS, we perform measurement experiments involving three different file sharing systems: BitTorrent (`bittorrent.com`), eDonkey (`edonkey2000.com`), and Gnutella (`gnutella.com`). At the time of our study, these were among the most popular file sharing systems in use on the Internet [4].^{2}

Most P2P file sharing systems have built-in mechanisms to discover existing users, which makes estimation of file sharing population relatively straight forward. The Gnutella system employs a decentralized approach to file

Table 3: Top-10 web service ASs (As of Sept. 2004)

R_{web}	ARIN		RIPE		APNIC	
	AS#	Name	AS#	Name	AS#	Name
1	3561	Savvis	8560	Schlund	3786	Dacom
2	2914	Verio	8220	Colt	4766	Korea Telecom
3	16509	Amazon	16276	Ovh	9304	Hutchison
4	21844	ThePlanet	3320	Deutsche Telekom	9318	Hanaro Telecom
5	11643	eBay	559	SWITCH	4808	Chinanet
6	13749	Everyones Internet	680	DFN	4134	China Telecom
7	7018	AT&T WorldNet	1273	C&W	2514	NTT
8	209	Qwest	702	MCI Europe	9848	GNG
9	701	UUNet	12312	Tiscali	4812	China Telecom
10	14134	Navisite	12322	Proxad	23880	Yahoo-KR

searching: individual Gnutella peers form an overlay network to propagate search messages. The eDonkey systems, on the other hand, relies on dedicated, centralized servers which peers must contact to search for a file. Similarly, BitTorrent relies on centralized *Trackers* from which a peer can obtain a list of other peers serving a particular file.

To estimate the population of BitTorrent, we downloaded about 2,800 *torrent files* from a well-known BitTorrent web site (`torrentspy.com`). A torrent file contains the meta data of a shared file, among which is the address of the BitTorrent *tracker* cognizant of peers sharing the file. Using BTtools (`bagley.org/~doug/project/bttools`), we obtained from each tracker a list of peers in possession of complete copies of the file (*seeds* in BitTorrent parlance). Over a period of four days, we collected about 634,000 distinct IP addresses of BitTorrent peers.

To estimate the population of eDonkey, we ran a toy eDonkey server and recorded the IP addresses of all the peers that contacted our server. eDonkey servers run a gossip protocol among themselves to maintain an up-to-date list of the server population. Each eDonkey peer maintains a list of servers and sends them periodic ping messages. In one day, our eDonkey server collected about 1,014,000 distinct IP addresses of eDonkey peers from these ping messages.

Finally, for the purpose of estimating the population of Gnutella, we ran a Gnutella client application on 20 PlanetLab nodes (10 in North America, 5 in Asia-Pacific, and 5 in Europe) and recorded the IP addresses of all Gnutella peers which exchanged traffic with us. Over a period of one week, we collected about 542,000 distinct IP addresses of Gnutella peers.

In total, we collected about 2.19 million distinct P2P IP addresses, which we subsequently mapped to their corresponding AS using BGP routing tables. Denoting by $U_{RA}(X)$ the utility of AS X as a residential Internet access provider, we computed this quantity for every AS X by counting the number of distinct P2P IP addresses associated with AS X. Of course, some P2P users perform more active downloads/uploads than others. By aggregating a sufficient number of IP addresses, we try to minimize any error that may be caused by ignoring such fine-

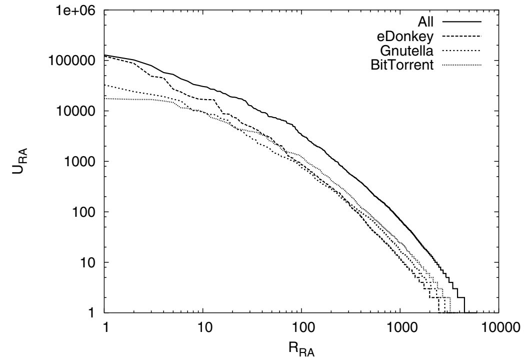


Figure 2: U_{RA} distribution

grained file sharing activity. We sort ASs by their utility $U_{RA}(\cdot)$ in decreasing order and assign them ranks, denoted $R_{RA}(\cdot)$. Table 4 lists the top-10 residential access ASs by region. In the European and Asia-Pacific regions, most of the high-ranking residential access providers are associated with telecom companies. In the North American region, retail Internet access business is more diversely distributed among telecom carriers and cable companies. Fig. 2 shows the entire R_{RA} vs. U_{RA} distribution, for all three P2P file-sharing applications individually as well as for their aggregate. In agreement with a recent study based on proprietary data set [20], $U_{RA}(\cdot)$ associated with the top 100 or so highest-ranking ASs can be characterized as a Zipf-type law with parameter -0.9.

One caveat in measuring P2P network usage is that the user base of different P2P systems is not uniformly distributed across residential networks [19]. Since several P2P applications with distinct features and evolving popularity coexist, relying on a single application may introduce sampling bias in capturing residential population. Our use of three popular file-sharing systems provides a reasonable coverage of the current file-sharing population. Our utility measurements can also be extended to not only cover other emerging P2P systems as they become popular, but also to include proprietary data such as per-AS statistics on residential subscriptions (including the amount of traffic generated by its connected residential customers), should the latter become publicly available.

Table 4: Top-10 residential access ASs (Oct. 2004–April 2005)

U_{RA}	ARIN		RIPE		APNIC	
	AS#	Name	AS#	Name	AS#	Name
1	1668	AOL	3320	Deutsche Telekom	4134	China Telecom
2	7132	SBC	3352	Telefonica España	4837	China Network
3	6478	AT&T WorldNet	3215	France Telecom	3462	HiNet
4	22909	Comcast Cable	12322	Proxad	4788	TMnet
5	577	Bell Canada	5617	Polish Telecom	1221	Telstra
6	22773	Cox	3269	Telecom Italia	4804	Microplex
7	812	Rogers Cable	5089	NTL	10091	SCV
8	7843	Adelphia	2856	BTnet	4812	Chinanet
9	6327	Shaw	6739	Cableuropa	9506	Magix
10	6128	Cablevision	3209	Arcor	7545	TPG

3.3 Business Access

Our approach to infer an AS's utility in providing business access relies on publicly available BGP routing tables to estimate the AS's bandwidth distribution. From a BGP routing table, one can infer provider-customer relationship among different ASs [10], and a naive estimate of an AS's bandwidth distribution would be the number of its customer ASs. However, this lump-sum measure does not distinguish between customer ASs of different sizes. A more meaningful measure of an AS's bandwidth distribution is the number of *downstream* ASs that are reachable from the AS, following the provider-customer relationship chains. A large transit customer AS with a high bandwidth requirement will then be properly weighted by its number of downstream customers. If a customer AS is multi-homed, i.e., obtaining its Internet access from several providers, it would typically impose lower bandwidth requirements on each provider than if it were single-homed. To infer $U_{BA}(X)$, the utility of AS X of providing business access, we assume that every AS has a unit bandwidth requirement. We then percolate each AS's bandwidth requirement up the provider-customer relationship hierarchy. When an AS is multi-homed, its per-provider bandwidth requirement gets divided by the number of its providers. We estimate $U_{BA}(X)$ in terms of the bandwidth distribution of AS X, computed as shown in Table 5.

Table 5: Computation of $U_{BA}(\cdot)$

```
for every AS $X$,
  unmark $X$.
  $U_{BA}(X) = 0$.
  $C(X)$ = # of $X$'s customer ASs.
  $P(X)$ = # of $X$'s provider ASs.
while (1)
  for each unmarked $X$ with $C(X) = 0$,
    mark $X$.
    for each provider $Y$ of $X$,
      $U_{BA}(Y) = U_{BA}(Y) + \frac{U_{BA}(X)+1.0}{P(X)}$
      decrement $C(Y)$ by one.
  if no AS has $C(\cdot) > 0$, exit.
```

We then sort those ASs with $U_{BA}(\cdot) > 0$ in decreasing order and assign them ranks, denoted $R_{BA}(\cdot)$. When different ASs have the same $U_{BA}(\cdot)$ value, we break ties by the size of the ASs' BGP-advertised address space. Table 6 lists the top-10 bandwidth reseller ASs. Most of them are associated with well-known tier-1 ISPs that operate continent-wide backbone networks. Note that in the European and Asia-Pacific regions, many top-ranking ASs are telecom companies.

We caution that our assumption of unit bandwidth requirement per AS may be too simplistic. For example, business customers which are not assigned public AS numbers are ignored in the computation of the $U_{BA}(\cdot)$ value. A more precise estimate of such intra-AS business customers could be obtained by examining intra-AS router-level connectivity, which, unfortunately, is not easy to discover from passive measurements. We rely here on an AS's address space size to partially account for the presence of such "hidden" customer ASs. Having access to proprietary information on intra-AS business customers would simplify this problem considerably.

3.4 Discussion

In the absence of readily available information about AS business models, our proposed methodology for inferring an AS's business model is based on the following assumptions: (1) ASs' web service utilities can be gleaned from the usage patterns of a popular web search engine, (2) utilizing widely adopted file sharing applications, an AS's residential access utility can be inferred from the size of its file sharing population, and (3) an AS's business access utility, as measured by its bandwidth distribution, can be estimated by counting its downstream AS customers. A careful study of the robustness of the proposed methodology to violations of these assumptions (e.g., a more URL-dependent network usage, non P2P-based residential access, a more cost-driven approach to providing business access) is necessary, but is left for future work.

Our measurement method identified about 40% of all BGP-advertised ASs as providing some form of Web hosting service, about 30% as providing residential access, and about 15% as providing business access.[3] The union of all identified ASs covers about 56% of all BGP-advertised ASs. Although close to 50% of all ASs are not categorized by our method, these are typically small ASs generating negligible traffic volume. For example, according to Net-

Table 6: Top-10 business access ASs (As of Sept. 2004)

R_{BA}	ARIN		RIPE		APNIC	
	AS#	Name	AS#	Name	AS#	Name
1	701	UUNet	1299	TeliaNet	4637	Reach
2	1239	SprintLink	702	MCI Europe	10026	ANC
3	3356	Level3	3320	Deutsche Telekom	2516	KDDI
4	7018	AT&T WorldNet	8220	Colt	3786	Dacom
5	209	Qwest	5511	France Telecom	703	UUNet AP
6	2914	Verio	1273	C&W	4766	Korea Telecom
7	3549	Global Crossing	6762	Telecom Italia	7474	Optus
8	3561	Savvis	3292	TDC	9225	Level3 AP
9	174	Cogent	6849	UKR Telecom	2764	Connect
10	3491	Beyond The Network	5400	BT Europe	7473	SingTel

Flow statistics obtained from a regional ISP,[4] the 56% identified ASs were responsible for 99% of all traffic observed by this ISP. This suggests that our methodology works well for the set of ASs that are responsible for the bulk of Internet traffic, but is of limited use for the many small ASs that contribute little to the overall traffic volume.

As the Internet continues to evolve, so does an AS's business model. However, since our proposed methodology relies mainly on being able to (1) identify the most "generic" business elements shared by existing ASs, and (2) infer each of these business elements by relying on appropriate "surrogate" measurements (or direct measurements, if available), we argue that our approach will remain applicable under changing Internet conditions (e.g., emergence of new "killer" applications), at least as long as generic business elements can be defined and viable surrogate measurements can be identified.

4 AS Business Characterization

From Tables 3, 4, and 6, one can see that some ASs rank very high with respect to more than one utility (e.g., Deutsche Telekom appears in all three), whereas other ASs rank high in only one category (e.g., Amazon in Web hosting, and Comcast in residential access). To compare these multi-variate AS utility profiles, and to associate each AS with one of the seven profiles listed in Table 1, we introduce the following quantitative metric. We first convert the ranks $R_{web}(X)$, $R_{RA}(X)$, and $R_{BA}(X)$ of an AS into their normalized counterparts, denoted by $r_{web}(X)$, $r_{RA}(X)$, and $r_{BA}(X)$, respectively. More specifically, we set $r_{web}(X) = R_{web}(X)/max\{R_{web}(i), i \in$ set of all ASs$\}$, so that $0 \leq r_{web}(X) \leq 1.0$. If $R_{web}(X)$ is unknown, we set it to $max\{R_{web}(i)\}$, reflecting our intuition that X has negligible web utility, so that in this case, $r_{web}(X) = 1.0$. Likewise for $r_{RA}(X)$ and $r_{BA}(X)$. We then define the *rank vector* $\mathbb{R}(X)$ corresponding to AS X as $\mathbb{R}(X) \triangleq (r_{web}(X), r_{RA}(X), r_{BA}(X))$. Note that $\mathbb{R}(X)$ can be interpreted as a point in the 3-dimensional hypercube, with the seven business models listed in Table 1 representing the extreme or corner points (i.e., $(0,0,0), (0,0,1), (0,1,0), \ldots, (1,1,0)$) of this hypercube, where 0 and 1 corresponds to "H" and "L" respectively. Intuitively, the business model of an AS X is determined by the minimal distance between its rank vector $\mathbb{R}(X)$ and the seven corner points. For example, as the rank vector $\mathbb{R}(X)$ gets closer to $(1,0,1)$, the business model of AS X is considered to be increasingly that of a residential access provider.

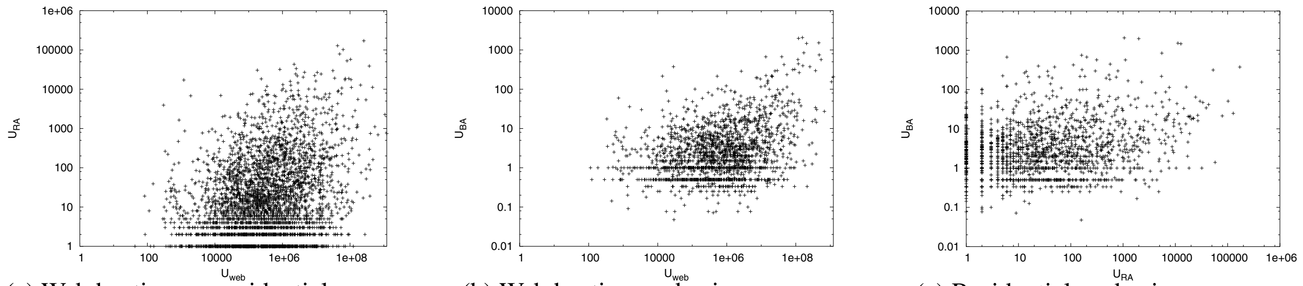
In Table 7, we list the top 10 North American ASs for each of the seven business models presented in Table 1. The top-10 ASs in the "Tier-1" category are those whose rank vectors are closest to the $(0,0,0)$ point in the 3D hypercube. Likewise for the remaining six categories. We observe that, first, ASs that are dominant in all three utility categories are indeed well-known tier-1 ASs. Second, in the "Network access" category where the primary utilities of the ASs are to provide Internet access to both business and residential customers, several of the high-ranking ASs are telecom companies. Third, the business profile of educational institutions falls under the "Retail service" category, where the primary utilities of an AS are to provide both Web hosting and residential access. Networks belonging to educational institutions usually host various academic web sites and at the same time provide Internet access to students living in university-owned housing. Fourth, several educational and research ASs are categorized under "Business access." These ASs serve purely as backbone networks connecting other smaller institutions' networks.

Next, we examine the correlation between the three utilities of an AS, for example, is an AS hosting a large volume of popular web content likely to serve a large number of residential customers as well? We use Kendall's rank correlation coefficient [12] to quantify these pairwise correlations. For samples $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ from a bivariate distribution, Kendall's (sample) τ coefficient is defined as $\sum_i \sum_{i<j} \binom{n}{2}^{-1} [sign(X_j - X_i) \cdot sign(Y_j - Y_i)]$, where $sign(x) = 1$ if $x > 0$, and -1 if $x < 0$. Kendall' τ provides a distribution-free measure of the strength of the association between two variables (i.e., monotonicity between two variables). The traditional Pearson product-moment correlation coefficient is less useful if Gaussian assumptions do not hold for the random variables at hand, as is the case for U_{web}, U_{RA}, and U_{BA}. To visualize the relationships between the three inferred utilities, we show in

Table 7: Top-10 ASs in different business categories (North American region)

Tier-1		Retail service		Business service		Network access	
AS#	Name	AS#	Name	AS#	Name	AS#	Name
3356	Level3	87	Indiana Univ.	14742	Internap	6383	BellSouth
7018	AT&T WorldNet	18566	Covad	297	NASA	6385	BellSouth
7132	SBC	1249	Univ. of Massachusetts	6922	Texas Backbone	13675	Verizon
209	Qwest	23504	Speakeasy	19782	Indiana Univ.	19158	USCarrier
1239	Sprint	2637	Georgia Tech.	5663	EDCnet	19752	Hydro One
3561	Savvis	14	Columbia Univ.	18695	Arbinet	22573	Northwestel
701	UUNet	25	UC Berkeley	5693	InteleNet	7776	Mebtel
852	Telus	4130	PSC	12179	N/A	25899	NOAnet
577	Bell Canada	18	Univ. of Texas	11588	El Dorado	7843	Adelphia
5650	ELI	20001	RoadRunner	4436	nLayer	10796	RoadRunner

Web hosting		Residential access		Business access	
AS#	Name	AS#	Name	AS#	Name
16509	Amazon	7757	Comcast	11537	UCAID
11643	eBay	20231	RoadRunner	6347	Savvis
14134	Navisite	13367	RoadRunner	1784	Global NAPs
8070	Microsoft	29737	WideOpenWest	6020	DCInet
7224	Amazon	27699	TSP	19151	IBIS7
11305	Interland	19115	Charter	2548	DIGEX
6432	Doubleclick	22269	Charter	3643	Sprint
26101	Yahoo	21508	Comcast	6509	Canarie
7859	Pair	11683	Earthlink	293	Energy Science Net
11443	OLM	4999	Sprint	2153	CSU



(a) Web hosting vs. residential access (b) Web hosting vs. business access (c) Residential vs. business access

Figure 3: Pairwise utility correlation

Table 8: Pairwise Kendall's τ for (U_{web}, U_{RA}, U_{BA})

Kendall's τ	ARIN	RIPE	APNIC	All
U_{web} vs. U_{RA}	0.1562	0.1540	0.1820	0.1540
U_{web} vs. U_{BA}	0.2332	0.1439	0.2032	0.1856
U_{RA} vs. U_{BA}	0.1864	0.2088	0.2483	0.2068

Fig. 3 pairwise scatterplots for U_{web}, U_{RA}, and U_{BA}. The graphs show that there exists non-negligible correlation between each pair of utilities.

Table 8 lists the resulting Kendall's (sample) τ values for pairwise correlations between U_{web}, U_{RA}, and U_{BA}. By calculating the pairwise correlation for the three geographic regions separately, we find that in the European and Asia-Pacific regions, the correlation between "residential access" and "business access" is higher than the other two pairwise correlations. As hypothesized earlier, this higher correlation may be due to dominance of the ISP market in these regions by incumbent telecom carriers. Since the ISP market in the Asia-Pacific countries, in particular, is highly regulated by the government, it is to be expected that the entire ISP business in this region is dominated by a few large telecom carriers. In the North American region, on the other hand, the correlation between "residential access" and "business access" is relatively low, reflecting a less regulated environment.

5 Inter-AS Traffic Demand Model

The proposed traffic demand model builds on our AS business characterization by using as key input the ASs' inferred utility profiles (i.e., U_{web}, U_{RA} and U_{BA}). We show how the new model can be used in conjunction with a given AS graph to generate realistic inter-domain traffic demand.

5.1 Modeling Framework

For describing inter-domain traffic demands, we postulate a general gravity model (see for example [23] and references therein), where traffic flow from body i to body j (denoted X_{ij}) is assumed to satisfy

$$X_{ij} \propto \frac{S_i \times T_j}{F_{ij}}, \qquad (1)$$

where:

- S_i: repulsive factor associated with "leaving" i,
- T_j: attractive factor, "approaching" j,
- F_{ij}: friction factor from i to j.

S_i, T_j, and F_{ij} are defined appropriately for each environment under study. For example, in applying the model to urban transportation networks, S_i and T_j typically represent the populations in areas i and j, with F_{ij} being in general a function of the distance between the two areas. When applying the model in the context of modeling intra-domain traffic demand, Zhang et al. [23] take S_i as the traffic volume entering at location i and T_j as the amount of traffic exiting at location j, and assume a common friction factor F_{ij} that does not depend on i and j [23]. They show that this model works remarkably well and yields traffic demand that is consistent with measured intra-domain traffic volumes. To put the gravity model to good use for describing inter-domain traffic demand, we need to define S_i, T_j, and F_{ij} within our context so as to reflect the specifics the Internet's AS environment.

Repulsive, attractive factors. We assume that a majority of inter-domain traffic in today's Internet can be attributed to two kinds of interactions: (i) communication between web servers and clients (called "web" traffic), and (ii) communication between two clients (called "inter-residential" traffic). Web surfing and media streaming belong to the first category, Email and file sharing belong to the second category.

A typical web transaction is an asymmetric two-way communication: client's request for web resource, and the corresponding response from the server. Thus, web traffic from AS i to AS j can be attributed to either a server in AS i returning web content requested by a client in AS j, or a client in AS i sending a request for web content served by a server in AS j. In the context of the gravity model, we model the volume of web traffic as a function of an AS's client population size and web content population size. Let $P_{RA}(X)$ be the client population size of AS X and $P_{web}(X)$ the web content population size of the AS. The volume of "web" traffic from AS i to AS j is then quantitatively expressed as the weighted sum of two products: $P_{web}(i)\cdot P_{RA}(j)+\kappa_w\cdot P_{RA}(i)\cdot P_{web}(j)$. The first term corresponds to the "response" traffic from AS i (of population size $P_{web}(i)$) to clients in AS j (of population size $P_{RA}(j)$). The second term corresponds to the "request" traffic from clients in AS i for web content in AS j. The parameter κ_w is the ratio of request traffic over response traffic, usually significantly less than 1.

The symmetric inter-residential traffic from AS i to AS j is modeled as $\kappa_r \cdot P_{RA}(i) \cdot P_{RA}(j)$. The parameter κ_r is a normalization factor that determines the relative weight of web traffic and inter-residential traffic. Combining web and inter-residential traffic, the total traffic volume from AS i to AS j can be estimated as: $P_{web}(i) \cdot P_{RA}(j) + \kappa_w \cdot P_{RA}(i) \cdot P_{web}(j) + \kappa_r P_{RA}(i) \cdot P_{RA}(j)$.

A key factor in specifying our traffic demand model concerns the issue of modeling the quantities $P_{web}(\cdot)$ and $P_{RA}(\cdot)$. However, as seen earlier in Section 4, the empirically derived quantities $U_{web}(\cdot)$ and $U_{RA}(\cdot)$ are in fact estimates of the population of web content and the population of clients, respectively, and are thus natural drivers of our gravity model. More generally, $P_{web}(i)$ and $P_{RA}(i)$ can be modeled as $f_1(R_{web}(i))$ and $f_2(R_{RA}(i))$, where $f_1(\cdot)$ and $f_2(\cdot)$ are monotonically decreasing rank-size functions; for example, in the case where $U_{web}(\cdot)$ follows roughly a type-1 Pareto distribution, $f_1(x) = x^{-\omega}$.

Friction factor. In urban transportation studies, the friction factor of the gravity model is typically a function of the distance between two regions. In estimating intra-domain traffic demand using the gravity model, Zhang et al. [23] assume a common, constant friction factor. In the inter-AS environment, such an assumption may not be realistic. For example, an over-provisioned path between two ASs may increase traffic flow between them, whereas an under-provisioned path is likely to decrease traffic flow between them.

Based on this observation, we define the friction factor between AS i and AS j as $R_{BA}(i,j)^{\beta}$, where $R_{BA}(i,j) = max\{R_{BA}(X) \mid X \in path(i,j), X \neq i,j\}$ and $path(i,j)$ denotes the set of transit ASs in the path between AS i and AS j. As defined in the previous section, $R_{BA}(X)$ is the rank of AS X among all business access providers. $R_{BA}(i,j)$ is thus the maximum rank of a transit AS between AS i and AS j. Assuming that an AS with higher transit rank is more likely to maintain a well-provisioned network, this definition of the friction factor captures the transit quality of the *bottleneck* AS of a given path. By tuning the parameter β, we can study the sensitivity of traffic demand as a function of the transit network quality (a smaller β means lower variability in transit quality).

Remarks: Note that the original gravity model postulates that interactions between nodes are *independent*. On the one hand, this assumption seems reasonable for modeling highly aggregated quantities such as inter-domain traffic flows, where the latter are, in general, sufficiently aggregated so that possible dependencies among finer-grained traffic flows can be safely ignored. On the other hand, by defining the friction factor in terms of $R_{BA}(\cdot)$, we may introduce subtle dependencies among inter-domain traffic flows, as $R_{BA}(X)$ is not independent of $R_{BA}(Y)$ if X is a downstream customer of Y, or vice versa. In this sense, the proposed "general gravity model" is not a gravity model in the strict sense, but allows for dependencies that may be genuine at the Internet's AS level, where inter-dependent traffic engineering is not uncommon.

5.2 Generation of Inter-AS Traffic Demand

Given an AS graph with N nodes and using the above gravity model, we can express the traffic demand from i to j as

$$X_{ij} \sim \frac{T_w(i,j) + \kappa_r \cdot T_r(i,j)}{R_{BA}(i,j)^{\beta}}, \qquad (2)$$

where

- $T_w(i,j) = f_1(R_{web}(i)) \cdot f_2(R_{RA}(j)) + \kappa_w \cdot f_2(R_{RA}(i)) \cdot f_1(R_{web}(j))$, and
- $T_r(i,j) = f_2(R_{RA}(i)) \cdot f_2(R_{RA}(j))$.

To produce the resulting inter-domain traffic demand matrix, we first *generate* for each node (AS X) in the graph its rankings in terms of the three utilities we identified ($\hat{R}_{web}(X)$, $\hat{R}_{RA}(X)$, $\hat{R}_{BA}(X)$). When generating these rank vectors, we must account for the pairwise correlation between the rankings as reported in Section 4. By definition, the ranking $\hat{R}_{BA}(\cdot)$ is determined solely by the topology of a given graph. Given a graph, $\hat{R}_{BA}(\cdot)$ can be computed independent of the other two rankings. Using $\hat{R}_{BA}(\cdot)$ as an *anchor*, we next generate $\hat{R}_{web}(\cdot)$ and $\hat{R}_{RA}(\cdot)$ based on a well-known method for generating multi-variate normal random numbers [18]. Our rank generation algorithm is described in Table 9. The input parameters to our algorithm are the AS graph and a 3×3 rank correlation matrix $\Sigma_\tau = \{\tau_{ij}\}$.

Table 9: Generation of $\hat{R}_{web}(\cdot)$, $\hat{R}_{RA}(\cdot)$, and $\hat{R}_{BA}(\cdot)$

```
Input: AS graph with $N$ nodes, $\Sigma_\tau = \{\tau_{ij}\}$
Algorithm:
// generation of $\hat{R}_{BA}(\cdot)$
Compute $U_{BA}(\cdot)$ by the method shown in Table 5.
Assign $\hat{R}_{BA}(\cdot)$ to ASs in a decreasing order of $U_{BA}(\cdot)$.

// generation of $\hat{R}_{web}(\cdot)$ and $\hat{R}_{RA}(\cdot)$
Convert $\Sigma_\tau = \{\tau_{ij}\}$ into $\Sigma_r = \{r_{ij}\}$,
    where $\Sigma_r$ is product moment correlation matrix
    with $r_{ij} = \sin(\frac{\pi}{2} \cdot \tau_{ij})$ (due to Kruskal [13]).
Compute a lower-triangular matrix $L$,
    such that $\Sigma_r = L \cdot L^T$.
Generate ($\hat{x_i}$, $\hat{y_i}$, $\hat{z_i}$) for each AS $i$,
    where $\hat{x}_i = \frac{\hat{R}_{BA}(i)}{N}$, and
    $\hat{y}_i$ and $\hat{z}_i$ = uniform random numbers $\in$ [0,1]
Obtain $(x_i, y_i, z_i)^T = L \cdot (\hat{x_i}, \hat{y_i}, \hat{z_i})^T$.
Assign $\hat{R}_{web}(\cdot)$ to ASs in a decreasing order of $y_i$.
Assign $\hat{R}_{RA}(\cdot)$ to ASs in a decreasing order of $z_i$.
Output: ($\hat{R}_{web}(X)$, $\hat{R}_{RA}(X)$, $\hat{R}_{BA}(X)$) of all ASs
```

6 Toward Model Validations

Recall that our empirical approach to determining the input data (i.e., utility profile-based AS business models) that drives the gravity model proposed in Section 5 is based exclusively on publicly obtainable/available data sets and does not use any actual traffic volume measurements. Thus a natural starting point for attempting to validate many aspects of our inter-domain traffic demand model is to gain access to traffic volume-related AS-specific data sets which are in general *not* publicly available. We follow this strategy by relying on a week's worth of (sampled) NetFlow measurements from a regional ISP. The data sets were collected from one of its access routers around the same time when we performed our own measurement experiments described in Section 3 (i.e., Oct. 2004). The captured traffic originates from or is destined to the ISP's networks, and the data sets contain, among other information, source/destination IP address prefixes of length at most 24 (due to anonymization), source/destination port numbers, and size of each traffic flow. We use these actual traffic measurements (i) to check a basic assumption underlying our gravity model, namely that the inter-domain traffic demand is determined by "web" and "inter-residential" traffic, (ii) to explore the adequacy of our key decision to use "surrogate" traffic measurements (e.g., data measuring the appearance of web content in search results, estimates for the number of P2P file-sharing users) instead of actual traffic volume estimates, and (iii) to provide a preliminary comparison between actual inter-domain traffic demand and those generated by our model.

6.1 Traffic Classification

To check the assumption explicit in Equation (2) that inter-AS traffic demand consists of the two components, "web" traffic and "inter-residential" traffic, we classify the flows in our NetFlow data set into "web" traffic and "inter-residential" traffic, using an up-to-date list of well-known port numbers. Given a traffic flow, if either source or destination port number is assigned to well-known web service (e.g., http, nntp, streaming), the flow is marked as "web" traffic. If either source or destination port number is associated with a well-known P2P file sharing application, the flow is marked as "inter-residential" traffic. However, an increasing number of applications do not use well-known port numbers, which makes it difficult to fully identify network traffic type based on port numbers alone. To improve upon the above naive traffic classification, for flows we cannot identify by source/destination port pair, we examine their source and destination address prefixes to heuristically infer its application type. More specifically, we randomly choose two IP addresses, one from each the source address prefix and destination address prefix, and perform a reverse DNS lookup. If either one of them has web-service related domain names (e.g., www* or web*), then we mark the flow as "web" traffic. If both IP addresses are resolved to well-known residential network domains (e.g., reshall.umich.edu or comcast.net), we mark the flow as "inter-residential" traffic.

Table 10 reports our traffic classification result. The reported percentages are based on total volumes of traffic. Although applying our heuristic reduces the amount of unknown traffic by 8%, uncategorized traffic still accounts for

Table 10: Inter-AS traffic classification

Classification	Web	Inter-residential	Unknown
Port-based	30.2%	28.8%	40.0%
Port-based + Heuristic	31.6%	36.0%	32.4%

one-third of all traffic, consistent with other available numbers [4]. We thus observe that with currently available traffic classification methods, our model appears to capture *at least* two thirds of actual inter-AS traffic. Improvements of state-of-the-art traffic classification techniques (e.g., [11]) can be expected to show a more accurate coverage of inter-AS traffic by our model.

6.2 Measurement Methodologies

Our methodology described in Section 3 for inferring an AS's utility profile and determining in turn its business model avoids on purpose actual traffic volume-related measurements. Instead, we rely on "surrogate" traffic measurements such as appearances of web content in search results or estimates of an AS's P2P file sharing population and assume that the latter are viable substitutes for the largely inaccessible actual traffic data. To check this assumption, we rely again on our NetFlow data sets and extract from them three distinct traffic volume measurements for each AS X: $T_{web}(X)$, $T_{RA}(X)$, and $T_{BA}(X)$, which correspond to "web-hosting" traffic, "residential access" traffic, and "business access" traffic, respectively. Using our classification of traffic in Section 6.1 into "web" and "inter-residential" traffic. we compute $T_{web}(\cdot)$ and $T_{RA}(\cdot)$, as described in Table 11.

Table 11: Computation of $T_{web}(\cdot)$ and $T_{RA}(\cdot)$

```
$T_{web}(\cdot)$ = $T_{RA}(\cdot)$ = 0 for every AS.
for each traffic flow $f$,
    if $f$ is web traffic,
        let $X$ = web hosting AS for $f$
        let $Y$ = client AS for $f$
        $T_{web}(X) = T_{web}(X) + size(f)$
        $T_{RA}(Y) = T_{RA}(Y) + size(f)$
    else if $f$ is inter-residential traffic,
        let $X$ = client AS 1 for $f$
        let $Y$ = client AS 2 for $f$
        $T_{RA}(X) = T_{RA}(X) + size(f)$
        $T_{RA}(Y) = T_{RA}(X) + size(f)$
```

The "business-access" traffic $T_{BA}(X)$ captures the volume of traffic going through AS X. To compute $T_{BA}(X)$, we use Gao's heuristics [10] to construct an AS-level routing path for each source-destination pair. We then increment $T_{BA}(\cdot)$ of every transit AS between the source-destination pair by the size of each flow.

Table 12: Pairwise Kendall's τ for (T_{web}, T_{RA}, T_{BA})

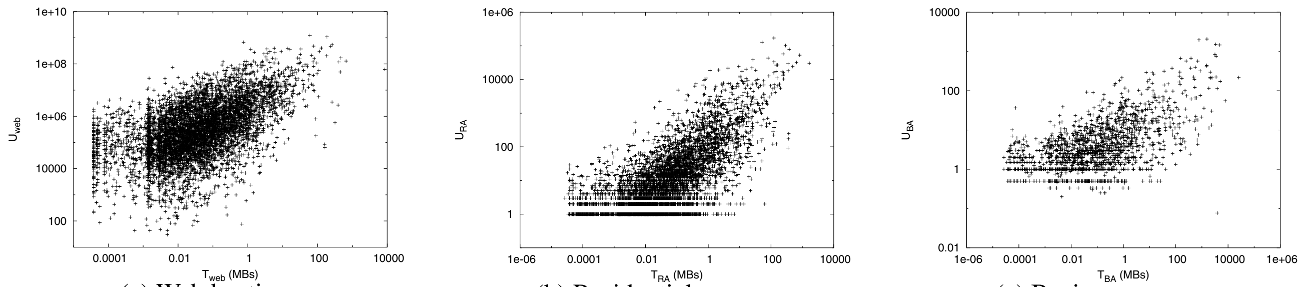
Kendall's τ	ARIN	RIPE	APNIC	All
T_{web} vs. T_{RA}	0.2490	0.1816	0.2752	0.2410
T_{web} vs. T_{BA}	0.1970	0.2467	0.2826	0.2440
T_{RA} vs. T_{BA}	0.1973	0.2489	0.2157	0.2371

Fig. 4 shows how well our inferred utilities (i.e., U_{web}, U_{RA} and U_{BA}) compare to their actual traffic-derived counterparts (i.e., T_{web}, T_{RA} and T_{BA}). We observe that in all three cases, ASs with high inferred utilities also have high measured traffic volumes. In Table 12, we quantify the pairwise correlation of T_{web}, T_{RA}, T_{BA}, just as we did earlier in Table 8 with the inferred utilities U_{web}, U_{RA}, and U_{BA}. Comparing Tables 8 and 12, we note that the pairwise correlation values are slightly underestimated by our methodologies, more so in the North American region than in the other regions. Overall, while the actual values differ, the generally low degree of pairwise correlations in T_{web}, T_{RA}, and T_{BA} is consistent with what we observed earlier for the inferred utilities U_{web}, U_{RA}, and U_{BA}, which suggests that our assumption of using appropriate "surrogate" traffic measurements instead of actual traffic measurements is not obviously unreasonable. However, there is clearly room for significant improvements.

6.3 Traffic Demand Model

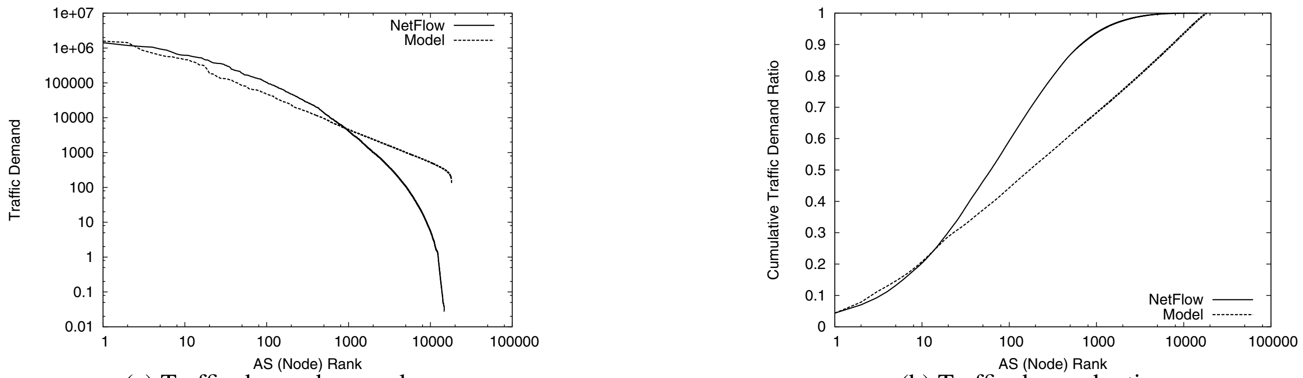
Ultimately, any inter-domain traffic demand model will be judged by how well the results compare to actual Internet data. In our attempt to provide such an initial comparison, we check whether our model, when combined with an inferred Internet AS graph and when appropriately parameterized, is capable of generating realistic traffic demand, consistent with actual demand measured in the Internet. To this end, we use an inferred Internet AS graph consisting of 18,221 nodes and 39,558 edges, and since our NetFlow data sets were collected from a single vantage point on the Internet, we compare model-generated traffic demand of a single source node against NetFlow traffic information, where the source node is chosen to be an AS S that has a business model comparable to that of the AS from which the NetFlow measurements were collected. We also present some preliminary result concerning the sensitivity of the generated inter-domain traffic matrix to the choice of model parameters.

Given the Internet-like pairwise rank correlation (i.e., τ_{ij} taken from the "All" column of Table 8), we assign ranks to each node based the method described in Section 5.2, considering the special case where the quantities $P_{web}(\cdot)$ and $P_{RA}(\cdot)$ are given by type-1 Pareto distributions, i.e., $P_{web}(X) = R_{web}(X)^{-\omega}$ and $P_{RA}(X) = R_{RA}(X)^{-\rho}$, with $\omega, \rho > 0$. Using this model, we then generate the traffic demand T_X that each node X maintains with our



(a) Web hosting (b) Residential access (c) Business access

Figure 4: Measured utilities vs. traffic



(a) Traffic demand vs. rank (b) Traffic demand ratio

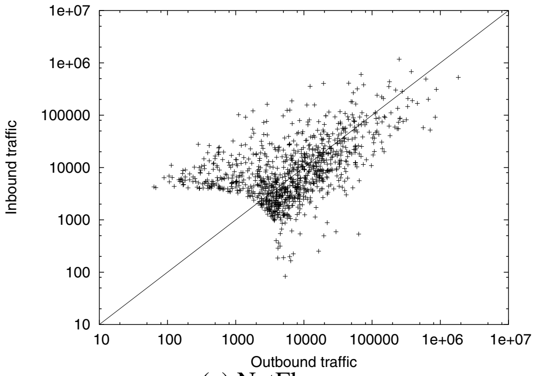
Figure 5: Single-source traffic demand comparison ($\omega = \rho = 1.0$, $\beta = 0.1$, $\kappa_w \in [0, 0.5]$, $\kappa_r \in [0, 1.0]$)

chosen source node S (i.e., $T_X = T_{SX} + T_{XS}$). At the same time, relying on the NetFlow data sets, we also obtain the measured traffic demand between each AS and the NetFlow collector AS, the regional ISP from which the NetFlow data was obtained.

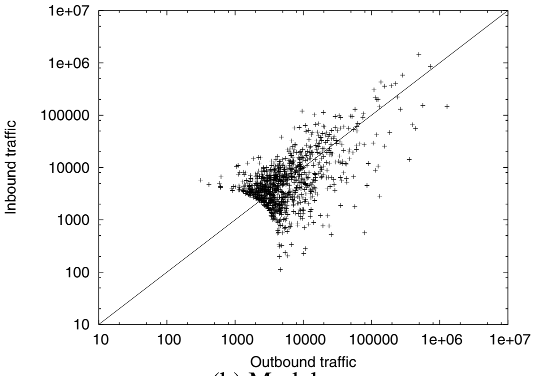
Fig. 5(a) compares model-generated traffic volumes with NetFlow-derived traffic volumes by showing volume vs. rank distributions.[5] In the case of the NetFlow-derived volume measurements, the traffic demand starts to fall steeply after rank 1000 or so, while the model-generated demand does not. This deviation is likely due to a deficiency of the type-1 Pareto distribution (used to model P_{web} and P_{RA}) as suggested by Figs. 1 and 2. In Fig. 5(b), we plot the cumulative traffic demand ratio; i.e., we examine what percentage of the total traffic demand the top-x percent ASs are responsible for. The actual traffic demand ratios of the top-ranking ASs (up to rank 30 or so) are matched well by our model, but for the same reason as before, the generated demand ratios start to deviate considerably from their actual counterparts when we include the lower-ranking ASs.

Finally, to compare model-generated and actual demand in terms of outbound and inbound traffic profiles, we plot in Fig. 6 for each node or AS X on the x-axis the volume of traffic from X to S, and on the y-axis the volume of traffic from S to X. Since we already know that our model is inadequate for predicting small ASs' traffic demand, we focus in the figure on the large ASs (i.e., top-1000 ASs in terms of their total traffic demand) to get an idea about how well our model predicts the inbound/outbound traffic for the critical large ASs. Fig. 6(a) shows the profile obtained from using NetFlow measurements, while Fig. 6(b) depicts the profile resulting from our model-generated interdomain demand. As the traffic profile of an AS moves further below the diagonal line, its business profile becomes increasingly that of a web service provider. Conversely, an AS whose traffic profile is located above the diagonal line is a typical residential access provider. Comparing the two profiles in Fig. 6, we see that the NetFlow-derived values are comparable to their model-generated counterparts, with less concentration around the diagonal, though.

To illustrate the effect of parameterization of our gravity model (2), we consider the parameter β. which determines how variable the transit quality of different networks is. A high value of β means that the transit quality of the higher-tier ASs is significantly better than that of the lower-tier ASs, and a low value of β implies a more uniform transit quality. In an extreme case where $\beta = 0$, the transit quality



(a) NetFlow



(b) Model

Figure 6: Single-source outbound vs. inbound traffic distribution ($\omega = \rho = 1.0$, $\beta = 0.1$, $\kappa_w \in [0, 0.5]$, $\kappa_r \in [0, 1.0]$)

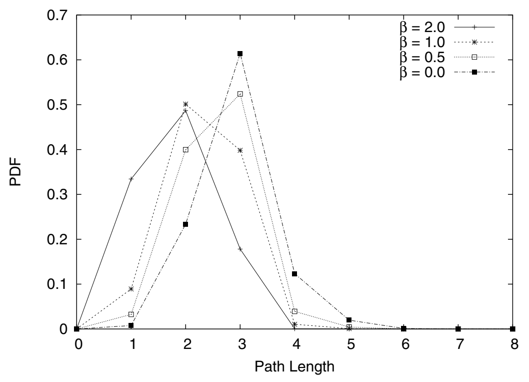


Figure 7: Topological distribution of traffic (ω = ρ = 1.0)

of all transit networks is considered the same. As described in Section 5, the friction factor F_{ij}, which is a function of β, reflects the transit quality of the path bottleneck and is thus a topological metric that depends on graph connectivity. To examine the effect of β in terms of its impact on the *topological distribution* of traffic demand, for a given path length l, let S_l denote the total sum of traffic exchanged between every pair of ASs that are distance l apart. We are interested in understanding how, depending on the parameter β, S_l changes with different path length l.

Fig. 7 shows the probability density functions of S_l for different values of β. As β increases, the entire density function is shifted to the left. Intuitively, in a high-β setting, the total traffic demand is more likely to be dominated by the demand between close-by source-destination pairs; demand between far-away pairs becomes negligible, (long paths are more likely to encounter a bottleneck than short paths and will be avoided). In short, a high-β setting imitates an Internet environment where traffic demand tends to be highly localized [22]. In contrast, in a low-β setting, traffic demand tends to be less sensitive to the bottleneck quality between source and destination, resulting in distant source-destination pairs exchanging non-negligible amount of traffic.

Besides the parameter β, we also examine the pairwise rank correlation matrix $\Sigma_\tau = \{\tau_{ij}\}$ and study the traffic profiles resulting from different degrees of rank correlation. As expected, increasing pairwise rank correlation results in higher correlation between outbound traffic associated with web hosting utility and inbound traffic attributed to residential access utility. Due to space limitation, we do not include the detailed results.

7 Conclusions

The Internet AS environment is a setting where establishing "ground truth" is notoriously difficult. For example, while it is relatively easy to infer AS maps of the Internet from publicly available BGP-derived data, the underlying measurements are known to provide only a very incomplete picture of Internet connectivity at the AS-level [5]. In turn, this creates significant challenges for accurately modeling the Internet's AS topology and large unresolved problems as far as validating the resulting models is concerned.

In this paper, we are concerned with an even more elusive aspect of the Internet's AS environment, namely the AS-level traffic matrix giving the traffic demand between any pair of connected ASs. For one, there exists no equivalent of the publicly available BGP-derived data, and this has led researchers to pursue a mostly model-based approach. Even worse, for fear of losing competitive advantage, ASs are very reluctant to provide any AS-related data. As a result, AS-specific traffic data is by and large not publicly available, causing researchers to look for "surrogate" measurements that are publicly available or obtainable (i.e., via measurement experiments that can be performed by anyone connected to the Internet) and that may shed some light on the nature of the actual inter-AS traffic demand. As far as model validation is concerned, this situation causes nightmares, because on top of examining the validity of a proposed model, it first requires checking that the considered surrogate measurements are indeed suitable and relevant as substitutes for the largely unavailable data.

By developing a flexible approach to generating inter-AS traffic matrices, we make four specific contributions:

1. Identification of relevant surrogate measurements that are publicly obtainable/available;
2. Derivation of AS-specific statistics from the measurements in 1 that are the key inputs to a general gravity model for inter-domain traffic demand;
3. Generation of inter-domain traffic demand from the gravity model in 2 that are not obviously inconsistent with actual demand;
4. Methodology for validating AS-level traffic demand models that puts to good use the few and rare proprietary data sets that some ISP have been willing to share with the networking research community.

While many of the specific details of our approach can be questioned and much room for improvements exist, we have demonstrated that overall, it is not only feasible, but also generates realistic inter-AS traffic demand with Internet-like characteristics. The chosen parameterization makes our model an attractive object for exploring "what-if" scenarios; by relating it to recent successful attempts at modeling intra-AS traffic demand, our model provides an initial framework within which one can start exploring the impact of intra-AS traffic engineering on inter-AS traffic engineering and vice versa.

References

[1] AGARWAL, S., NUCCI, A., AND BHATTACHARYYA, S. Towards Internet-Wide Network Management. In *Proc. of IEEE Infocom 2005 Poster Session* (March 2005).

[2] AKELLA, A., CHAWLA, S., KANNAN, A., AND SESHAN, S. On the Scaling of Congestion in the Internet Graph. *ACM SIGCOMM Computer Communication Review 34*, 3 (July 2004).

[3] AKELLA, A., MAGGS, B., SESHAN, S., SHAIKH, A., AND SITARAMAN, R. A Measurement-Based Analysis of Multihoming. In *Proc. of ACM SIGCOMM* (Aug 2003).

[4] CACHELOGIC, INC. The True Picture of Peer-to-Peer Filesharing. http://www.cachelogic.com.

[5] CHANG, H., GOVINDAN, R., JAMIN, S., SHENKER, S. J., AND WILLINGER, W. Towards Capturing Representative AS-level Internet Topologies. *Computer Networks 44*, 6 (2004).

[6] FANG, W., AND PETERSON, L. Inter-AS Traffic Patterns and Their Implications. In *Proc. of IEEE Global Internet Symposium* (1999).

[7] FEDERAL HIGHWAY ADMINISTRATION. Calibrating and Testing a Gravity Model for Any Size Urban Area, August 1983. U.S. Department of Transportation.

[8] FELDMANN, A., GREENBERG, A., LUND, C., REINGOLD, N., REXFORD, J., AND TRUE, F. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. *IEEE/ACM Transactions on Networking 9*, 3 (June 2001).

[9] FELDMANN, A., KAMMENHUBER, N., MAENNEL, O., MAGGS, B., PRISCO, R. D., AND SUNDARAM, R. A Methodology for Estimating Interdomain Web Traffic Demand. In *Proc. of ACM Internet Measurement Conference* (2004).

[10] GAO, L. On inferring autonomous system relationships on the Internet. In *Proc. IEEE Global Internet Symposium* (Nov 2000).

[11] KARAGIANNIS, T., PAPAGIANNAKI, D., AND FALOUTSOS, M. BLINC: Multilevel Traffic Classification in the Dark. In *Proceedings of ACM SIGCOMM '05* (2005).

[12] KENDALL, M. G. *Rank Correlation Methods.* Charles Griffin & Company, 1962.

[13] KRUSKAL, W. Ordinal Measures of Association. *Journal of the American Statistical Association 53* (1958).

[14] MACHIRAJU, S., AND KATZ, R. H. Verifying Global Invariants in Multi-Provider Distributed Systems. In *Proc. of HotNets-III* (2004).

[15] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Towards Coordinated Interdomain Traffic Engineering. In *Proc. of HotNets-III* (2004).

[16] MEDINA, A., TAFT, N., SALAMATIAN, K., BHATTACHARYYA, S., AND DIOT, C. Traffic Matrix Estimation: Existing Techniques and New Directions. In *Proc. of ACM SIGCOMM* (2002).

[17] PÖYHÖNEN, P. A tentative model for the volume of trade between countries. *Weltwirtschaftliches Archive 90* (1963).

[18] RUBINSTEIN, R. Y. *Simulation and the Monte Carlo Method.* John Wiley & Sons, 1981.

[19] SANDVINE, INC. Regional characteristics of P2P - File sharing as a multi-application, multi-national phenomenon, 2003. White paper.

[20] SEN, S., AND WANG, J. Analyzing Peer-To-Peer Traffic Across Large Networks. *IEEE/ACM Transactions on Networking 12*, 2 (April 2004).

[21] TANGMUNARUNKIT, H., GOVINDAN, R., JAMIN, S., SHENKER, S., AND WILLINGER, W. Network Topology Generators: Degree-Based vs. Structural. In *Proc. of ACM SIGCOMM* (2002).

[22] UHLIG, S., BONAVENTURE, O., MAGNIN, V., RAPIER, C., AND DERI, L. Implications of the Topological Properties of Internet Traffic on Traffic Engineering. In *Proc. of ACM Symposium on Applied Computing* (2004).

[23] ZHANG, Y., ROUGHAN, M., DUFFIELD, N., AND GREENBERG, A. Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads. In *Proc. of ACM SIGMETRICS* (2003).

Notes

[1]Recent years have seen accelerated migration of dialup modem users to broadband subscription and, according to some recent statistics, DSL subscription in the US nearly doubled in 2003 (`point-topic.com`).

[2]Since traffic on the FastTrack/KaZaA network has declined sharply in the past few years and continues to decline, we did not include it in our study.

[3] The collected measurement data sets are available at `http://topology.eecs.umich.edu/traffic/`.

[4]We thank Manish Karir for making the NetFlow data available to us.

[5]The random parameterization of κ_w and κ_r reflect AS-dependent variations of traffic components.

Geographic Locality of IP Prefixes

Michael J. Freedman
New York University
mfreed@cs.nyu.edu

Mythili Vutukuru, Nick Feamster, Hari Balakrishnan
Massachusetts Institute of Technology
{mythili,feamster,hari}@csail.mit.edu

ABSTRACT

Information about the geographic locality of IP prefixes can be useful for understanding the issues related to IP address allocation, aggregation, and BGP routing table growth. In this paper, we use traceroute data and geographic mappings of IP addresses to study the geographic properties of IP prefixes and their implications on Internet routing. We find that (1) IP prefixes may be too coarse-grained for expressing routing policies, (2) address allocation policies and the granularity of routing contribute significantly to routing table size, and (3) not considering the geographic diversity of contiguous prefixes may result in overestimating the opportunities for aggregation in the BGP routing table.

1. Introduction

Today's Internet routing infrastructure achieves scalability by expressing reachability for large groups of IP addresses using a single IP prefix in a route advertisement. Today's largest Internet routing tables provide reachability to hundreds of millions of end hosts with nearly 200,000 routes [5]. IP addresses that are nearby in IP space may be geographically or topologically diverse, and vice versa. This paper *quantifies* this lack of correspondence. Information about the geographic location of hosts within IP prefixes can also help us better understand many issues related to IP address aggregation and allocation and their effect on BGP routing table growth.

Our study uses extensive traceroutes and leverages IP-to-geographic mapping techniques to examine the geographic properties of multiple destinations *within a single prefix*. Our dataset includes traceroutes to at least 4 IP addresses within each prefix of the global routing table, as well as traceroutes to 1.6 million unique Web clients and servers that exchanged content over CoralCDN, a popular peer-to-peer content distribution network [3].

Towards this goal of understanding the geographic properties of IP prefixes, this paper makes three findings. First, an IP prefix may express only very coarse geographic information about the destinations (and networks) that it comprises. This property of the geographic diversity of hosts within a prefix is important for techniques that assume that hosts within an IP prefix are topologically close. As expected, we find that "shorter" IP prefixes, which represent a larger portion of the IP address space, tend to comprise destinations in a large number of geographic locations, spread over long distances. For example, more than half the prefixes with mask lengths between 8 and 15 span a distance of more than 100 miles. More surprisingly, we find that "longer" prefixes, albeit a small fraction of them, can be quite geographically diverse: about 1.4% of the prefixes with mask lengths between 24 and 31 span a distance of more than 100 miles, and some /24 prefixes span distances of more than 10,000 miles!

Second, autonomous systems (ASes) commonly advertise multiple discontiguous IP prefixes for networks in the *same* geographic location. In this case, the Internet routing table must carry multiple routes for a group of destinations in a single geographic location and a single AS, because the addresses cannot be expressed as a single IP prefix. This finding suggests that an Internet routing infrastructure whose routing granularity more closely reflects geography could significantly reduce the size of the global routing tables. Additionally, fragmented address allocation explains 65% of the cases where a single AS was advertising discontiguous prefixes from the same location, which suggests that IP address renumbering could significantly reduce the size of the BGP routing table.

Finally, ASes sometimes announce contiguous prefixes from *different* geographic locations. Ongoing studies, such as the CIDR Report [2], presume that all contiguous prefixes originated by an AS should be aggregated into a single IP prefix. However, these studies do not consider whether these prefixes actually represent geographically diverse networks that are intentionally represented as separate routes. By ignoring location information, the CIDR Report may overestimate the opportunities for aggregation by a factor of three.

2. Related Work

Padmanabhan *et al.* [9] develop a set of techniques to map IP addresses to geographic locations. One of their techniques "clusters" IP addresses at the granularity of an IP prefix to map them to a location. The authors observe that the accuracy of their method in mapping an IP address is related to the geographic spread of the hosts within the prefix containing that IP address. Our work aims to gain a deeper understanding of geographic diversity of the hosts within a single IP prefix.

The geographic locality of IP prefixes is significant for systems like Network Aware Clustering (NAC) [6], which group hosts that belong to the same prefix of the BGP routing tables into clusters, which are used in applications like content distribution and proxy positioning. These clustering schemes rely on the assumption that hosts within a prefix are likely to be topologically close and under the same administrative domain. We investigate the validity of this assumption in Section 4.1.

Earlier work has also studied impact of factors like IPv4 address allocation and aggregation on the growth of the BGP routing table [1, 7]. Bu *et al.* [1] find that address fragmentation (where a set of prefixes originated by an AS *cannot* be summarized by one prefix) is the biggest factor contributing to BGP routing table growth. Our study also reveals many instances where an AS announces discontiguous prefixes, even from the *same* geographic location.

The CIDR Report studies contiguous prefixes announced by the same AS and the missed opportunities for aggregation by ASes [2]. In our study, we find that contiguous prefixes announced by the same AS are sometimes geographically far apart; aggregating such prefixes might conflict with an AS's traffic engineering or load balancing goals. Thus, the aggregation opportunities suggested by the CIDR Report might not all be feasible.

3. Data

This paper uses three datasets generated by traceroute measurements to study the relationship between IP prefixes and locality. We mapped IP addresses to IP prefixes using longest-prefix matching on a BGP table from RouteViews [8] from February 27, 2005. This table had approximately 170,000 IP prefixes.

As shown in Table 1, *Clients* and *Servers* refer to traceroutes taken to Web clients and servers that exchanged content over CoralCDN, a peer-to-peer content distribution network that receives approximately 10 million HTTP requests per day from widely-dispersed clients [3]. The client traces cover a 14-day period starting on February 13, 2005, while the server trace covers a single day (April 26, 2005). Each CoralCDN Web proxy—there are approximately 225 such proxies deployed on PlanetLab [10]—performed a traceroute to every client *destination* IP.

While these CoralCDN datasets provide a workload corresponding to a real user population, we also sought to provide coverage of all IP prefixes from the RouteViews table. For the *Breadth* dataset, we performed traceroutes to 4 uniformly distributed IP addresses per advertised prefix, using 25 PlanetLab hosts as sources. Note that these traceroutes traverse IP addresses from multiple prefixes.

Dataset	Period	Traceroutes	Destinations	IPs	Prefixes
Clients	Feb 13-27, 2005	6,565,844	1,599,228	692,080	45,573
Servers	Apr 26, 2005	71,621	36,387	64,378	9,589
Breadth	Apr 25, 2005	675,797	649,441	246,626	161,974

Table 1: Traceroute datasets. The last two columns show *reachable* IP addresses and prefixes: routers and destinations from which ICMP replies were received.

Thus, *Breadth* actually includes many more data points than four per prefix, especially for transit ASes.

Dataset	Mapped	Inherited	Prefixes	ASes	Locations
Clients	313,573	180,487	6,136	1,244	1,363
Servers	22,749	5,032	1,693	541	748
Breadth	176,601	130,621	6,828	1,605	1,206

Table 2: IP-to-location assignments.

We use the RouteViews table to map IP addresses to their ASes and DNS naming heuristics to map IPs to locations, as described in Section 3.1. Table 2 characterizes the number of IP addresses *mapped* to an AS number and a location (at the city level). We call this location *inherited* if the destination is not reachable itself (whereupon we assign it to the location of its closest reachable upstream router instead). The *inherited* dataset is a subset of *mapped*, which in turn is a subset of the *destination* IPs in Table 1. Table 2 also shows the total number of unique IP prefixes, ASes, and locations in each dataset.

3.1 Mapping IP addresses to locations

We use `undns` [11] to map IP addresses to locations. `undns` extracts geographic information from a DNS name, which is useful because network operators often use geographically meaningful names for routers. For example, a DNS name of the form *qwest-gw.n54ny.ip.att.net* refers to an AT&T (AS 7018) router peering with Qwest, located at an exchange point on 54th street in New York City. Other studies have also used this approach [9].

Unfortunately, naming heuristics vary between ISPs, and parsing is a manual process. ISPs may name routers by city name or code, airport code, or some 4-to-6 letter abbreviation for city and state. In addition, ISPs incorporate such information in hostnames differently; even a single AS may use multiple heuristics. For example, Verio (AS 2914) names gateways in one manner (*e.g.*, *att-gw.nyc.verio.net*) and customer addresses in another (*e.g.*, *vl-101.a02.nycmny03.us.ce.verio.net*). Router names can also be ambiguous: for example, *nycmng-washng.abilene.ucaid.edu* is located in New York but peers with a router in Washington, D.C. In such special cases, we manually pinged routers from diverse locations to better understand their ISP-specific naming heuristics.

`undns` version 0.1.27a includes manually written hostname parsing rules for 247 ASes, mostly Tier-1 and Tier-2 ISPs in the US and Europe. We added support for 169 additional ASes (including smaller ISPs) and expanded the tool's international coverage. The latter is especially important for the *Clients* dataset, which includes significant amounts of traffic from Asia. We spot-checked location estimates after running `undns` for some IP addresses in known locations.

Given a city-level location estimate for a particular IP address, we also assign to it the latitude and longitude coordinates for that city, which allows us to estimate the distance between two IP addresses.

3.2 Limitations of mapping technique

Our data has several limitations. First, a reverse DNS mapping from IP address to hostname may not exist; such records existed for only 50%-60% of all *unique* reachable IP addresses. Second, `undns` may not have a parsing rule to map the hostname to a location; our ruleset assigned locations to about one-third of known hostnames. Third, `undns` may return incorrect IP-to-AS number mappings. Finally, some destinations were not reachable via traceroute. We now discuss mitigating factors for the first two limitations and solutions for the latter two.

While we could resolve the hostnames of less than 60% of IPs, we found that internal ISP routers—as opposed to gateway routers or customer addresses—were more commonly missing reverse DNS records. These routers are unlikely to express more geographic diversity than that already captured by gateways and customers, so this limitation should not significantly affect our results.

Even though `undns` assigned locations for only one-third of all *unique* hostnames, two factors reduced the impact of this poorer coverage. First, our ruleset provides very good coverage for real-world traffic patterns, as we supply more detailed rules for popular ASes. In fact, we resolved the location of 90% of probed IPs in *Servers* (*i.e.*, when counting *all* instances, instead of only unique instances, of hostnames). Second, the hostnames that had no locality information were most commonly at the network edges where dynamic addressing is used (*e.g.*, cable modem, DSL, and dialup connections). This may inflate the number of hosts with unassigned locations.

`undns` uses the hostname of an IP address to determine its AS number, which could cause us to mistakenly believe an ISP is announcing a discontiguous prefix. For example, an IP address in AS 6395 (Broadwing Communications) carries the hostname suffix *.northwestern.edu*, even though its corresponding /14 prefix is announced solely by Broadwing, which provides transit service for Northwestern University (AS 103). To solve this problem, we assigned an AS number to an IP address by performing longest-prefix matching against the RouteViews table.

Finally, many destinations were not directly reachable when performing traceroutes: 57% of addresses in *Clients*, 52% in *Servers*, and 76% in *Breadth*. This limitation is generally due to firewalls blocking ICMP packets at large portions of the networks' edges. and many destinations in *Breadth* were unused IP addresses. To solve this problem, we assigned an unreachable destination IP address to the location of its last reachable upstream router. Our use of traceroutes enables us both to discover routable IP addresses for firewalled or unused destinations and to determine the upstream addresses for inherited locations.

4. Results

We first examine the geographic diversity of individual IP prefixes, paying particular attention to the *maximum* geographic distance between any two pairs of IP addresses within a single prefix. We then study the extent to which a single AS advertises multiple discontiguous prefixes that refer to endpoints at a single location, as well as the causes of these advertisements. Finally, we study the extent to which an AS advertises contiguous prefixes for hosts in diverse geographic locations.

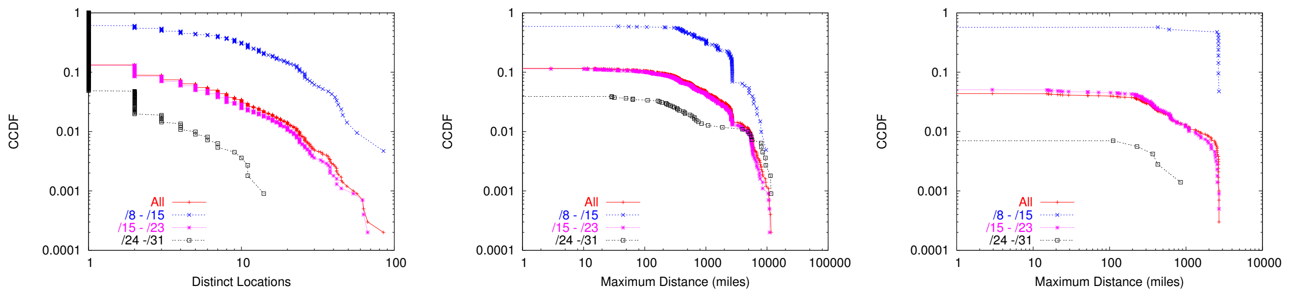
4.1 Single prefix with multiple locations

In this section, we study the extent to which a single IP prefix comprises hosts in multiple geographic locations (thus potentially obscuring potentially useful information by over-aggressive aggregation). Figure 1(a) shows the number of distinct geographic locations contained within a single geographic prefix for the *Clients* dataset. As expected, shorter prefixes tend to comprise more geographic locations.

Figure 1(b) shows that, not only do the shorter prefixes span more geographic locations, but these hosts also span a much wider geographic distance: nearly half of the prefixes in the /8-/15 range span a distance of more than 100 miles. Several of the prefixes in this range are either European backbones or broadband access providers in the United States: for example, from the *Clients* dataset, we find that AS 7132 (SBC) advertises a single /16 that contains 64 distinct locations spread across the United States. Transit ASes with smaller address allocations also advertised prefixes containing geographically diverse hosts: *e.g.*, AS 7657 (The Internet Group, a New Zealand ISP), advertised a /24 whose IP addresses span 1,400 miles.

Because ASes (particularly US-based backbone ISPs) often allocate sub-prefixes from a single large IP prefix, we expected that prefixes that are allocated to transit ISPs are more likely to have geographically diverse prefixes than those that are allocated to ASes that do not transit traffic for others. As shown in Figure 1(c), roughly 97% of all prefixes announced by stub ASes (and more than 99% of all prefixes in the /24-/31 range announced by stub ASes) were announced from the same location.[1] The remain-

[1]Classifying an AS as a "stub" turns out to be difficult, as acquisitions, unorthodox transit relationships (*e.g.*, Harvard University appears as a transit for MIT in RouteViews), etc., preclude classifying the leaves of the RouteViews graph as stub ASes. Instead, we classify an AS as a stub



(a) Number of Distinct Locations (b) Maximum Distance (c) Maximum Distance for Stub ASes only

Figure 1: Geographic diversity of IP addresses *within a single prefix*. Graphs show *complementary* CDFs for the *Clients* dataset; other datasets exhibit similar properties.

ing prefixes announced by stub ASes, however, may contain locations that span large distances. For example, AS 6316 (StarNet) advertises a single /18 that contain hosts spanning over 2,000 miles in 9 locations. Another striking example is AS 4637 (Reach, an Asia-Pacific backbone "with direct connectivity to the US and Europe"), which advertises several /24 prefixes spanning over 10,000 miles (such as 202.84.142.0/24, which contains hosts in Perth, Australia and Dallas, Texas)!

About half of prefixes in the /8-/15 range contain IP addresses in multiple geographic locations, and about 97% of both prefixes longer than /24 and prefixes announced by stub ASes refer to IP addresses in only a single geographic location, which is expected. When stub ASes do advertise prefixes that contain hosts in different geographic locations, however, it is often the case that these hosts are not close together at all.

We hypothesized that, because large prefixes exhibit geographic diversity, large ASes might exhibit similar geographic diversity. That is, ASes with high degree (according to the RouteViews table) might announce prefixes from many diverse geographic locations. Interestingly, there are many small ASes that nevertheless announce geographically diverse prefixes as well: the correlation coefficient between AS degree and maximum distance between IP addresses contained within that AS is only 0.07, and many ASes with small degree commonly contain geographically diverse hosts. For example, AS 6509 (Canarie Inc., Canada), a relatively small organization with an outdegree of only 38 in the RouteViews table, announces a prefix 205.189.32.0/24 that spans locations that are 2,300 miles apart.

4.2 Discontiguous prefixes with single location

In this section, we analyze how frequently discontiguous prefixes (which cannot be aggregated) are announced by an AS from the same geographic location. We found that discontiguous prefixes formed between 70% and 74% of the total number of prefixes mapped in the three datasets. Discontiguous prefixes from the same geographic location and AS indicate that an IP prefix is too fine-grained.

Cause	Clients	Servers	Breadth
Fragmented Allocation	65.8	82.5	59.0
Load balance	1.5	1.9	3.9
Misclassification	4.5	4.8	13.8
Unknown	28.2	10.9	23.3

Table 3: Analysis of the possible causes for the presence of discontiguous prefixes from the same geographic location within an AS.

Table 3 summarizes possible reasons for ASes announcing discontiguous prefixes from the same location, as well as their relative frequencies in our three datasets. Fragmented allocation is the single biggest reason for discontiguous prefixes being announced from the same AS and location: 65% of the discontiguous prefixes that appear in the routing table result from regional routing registries allocating discontiguous prefixes to ASes. We now analyze the causes for discontiguous prefixes in greater detail.

4.2.1 Fragmented allocation

IPv4 addresses are allocated by four Regional Internet Registries (RIRs): APNIC (Asia Pacific), ARIN (North America), LACNIC (South America and the Caribbean), and RIPE (Europe, Central Asia, and the Middle East).[2] The registries publish information on every block of IP space allocated by them. A typical allocation appears as:

```
arin|US|ipv4|19.0.0.0|16777216|19880615|assigned
```

This record specifies that a block of 16,777,216 contiguous addresses (*i.e.*, a /8) beginning from IP address 19.0.0.0, had been assigned to an organization on June 15th, 1988.

if it has fewer than 5 downstream "customer" ASes per the classification algorithm from Gao [4].

[2]In February 2005, a fifth RIR (AfriNIC) began full operation, covering registration for Africa. However, our datasets included the older registrations managed by ARIN and RIPE.

Registry	% fragment	% discontig	% all	% used
APNIC	25.11	31.90	30.97	81.07
ARIN	43.69	30.00	27.30	85.97
LACNIC	5.70	14.99	15.89	68.49
RIPENCC	25.50	23.11	25.85	86.38

Table 4: Contribution of the various registries (*Breadth* dataset).

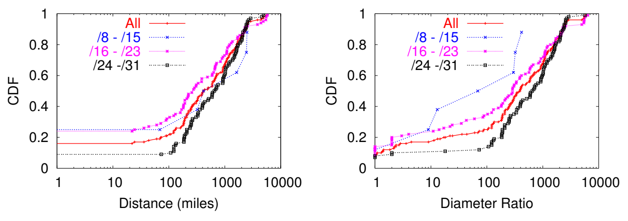
Using such allocation records, we investigated how often fragmented allocation was the cause for ASes announcing discontiguous prefixes. If a pair of discontiguous prefixes are from discontiguous allocations, then we conclude that an fragmented allocation has occurred.

Table 4 gives a registry-wise breakdown of the prefixes from fragmented allocations, discontiguous prefixes and the total number of prefixes observed. We have also tabulated the total fraction of the address space allocated at these registries. The table shows that LACNIC experiences less allocation pressure and similarly causes fewer fragmented allocations.

To further understand the reasons behind discontiguous allocations, we examined the allocation patterns of the 20 $\langle$AS,location$\rangle$ pairs in *Breadth* from which the largest number of discontiguous prefixes originated. We observed that 23% of the discontiguous allocations in these 20 $\langle$AS,location$\rangle$ pairs were made from discontiguous spaces *on the same day*, indicating that the registries were forced to make such assignments due to the paucity of IPv4 addresses. The remaining 77% of the allocations were made during different periods of time. Possible explanations for discontiguous address space allocations to an AS at different points of time are: (1) scarce IPv4 addresses are allocated conservatively to organizations, resulting in a fragmented set of addresses for each organization; and (2) two or more organizations with discontiguous addresses have one AS number due to a merger or acquisition.

4.2.2 Load balance

An AS might announce a specific subnet of a bigger prefix in order to balance load over its two incoming links. For example, consider an AS with prefix p_i and two incoming links L_1 and L_2, which desires that the traffic to a more specific (*i.e.*, "longer") prefix p_j arrive through link L_1 and the remaining traffic through link L_2. To achieve this goal, it announces the "longer" prefix p_j over link L_1 and p_i over L_2. This practice is commonly referred to as "BGP hole punching". Let $D_{discontig}$ denote the set of all discontiguous prefixes in a dataset. To determine whether a pair of prefixes $\{p_i, p_j\}$ appears in $D_{discontig}$ due to hole punching, we check if their AS announces a supernet p_s that contains both p_i and p_j from the same location, thus producing a discontiguous pair of prefixes. We can observe from Table 3 that the number of discontiguous prefixes that appear due to load balancing is negligible—between 1.5% and 3.9% of the total number of discontiguous prefixes.



(a) Maximum Distance (b) Diameter Ratio

Figure 2: Geographic diversity of *contiguous prefixes announced by the same AS*. Graphs are for the *Breadth* dataset; other datasets show similar results.

4.2.3 Misclassification

As our location mapping data is incomplete, we could have misclassified a set of contiguous prefixes as discontiguous due to the absence of traceroutes to some prefixes. Consider a set of contiguous prefixes $\{p_i, p_j, p_k\}$. Assume that we have mapped p_i and p_k to a location L, but we do not have any location for prefix p_j. Then, by observing only prefixes p_i and p_k, we might mistakenly assume that the AS is announcing discontiguous prefixes from the same location. Hence, for every pair of discontiguous prefixes $\{p_i, p_k\} \in D_{discontig}$, we check if the "missing" intermediate prefixes are in fact announced by the AS in the RouteViews table. If so, we count this as an instance of misclassifying the pair $\{p_i, p_k\}$ as discontiguous.

In Table 3, we observe that the *Breadth* dataset has more misclassifications than the other two. This result can be explained by the fact that, despite tracerouting to all advertised prefixes, we could not map all prefixes' locations due to the limitations of `undns`. This limitation has a stronger influence on *Breadth* (which reached 161,974 prefixes) than on *Clients* (which reached 45,573).

4.3 Contiguous prefixes with multiple locations

In this section, we study the extent to which ASes advertise contiguous IP prefixes that refer to networks in diverse geographic locations. We found 2,281 pairs of contiguous prefixes advertised by 384 different ASes. Of these pairs of prefixes, about one-fourth (607) of the pairs contained hosts in distinct geographic locations.[3] This finding suggests that the opportunities for aggregation may be less than that implied by the CIDR Report.

Figure 2(a) shows a CDF of the maximum distance spanned by hosts contained within a set of contiguous pre-

[3]Note that this measure is also a lower bound, as certain IP prefixes that we attributed to the same location might actually contain hosts in a different location that we did not probe.

fixes advertised by the same AS.[4] About 10% of all sets of contiguous prefixes were advertised from a single geographic location.

To better understand whether or not it makes sense to aggregate two contiguous prefixes, we defined a metric called the *diameter ratio* that highlights cases where a pair of contiguous prefixes represent two well-defined geographic clusters that are significantly far apart from each other. The *diameter ratio* is defined formally as follows:

$$\text{diameter ratio} \quad = \quad \frac{\text{maxdist}(L_1 \cup L_2)}{\min(\text{maxdist}(L_1), \text{maxdist}(L_2))}$$

where L_i is the set of locations contained in prefix p_i and $maxdist$ is the maximum geographic distance between any pair of IP addresses in a set of IP addresses (*i.e.*, the "diameter" of the prefix). When either L_1 or L_2 contains only a single location, we set the denominator to 1. Intuitively, the diameter ratio is large when the locations within each of one or both of two prefixes are close together, but the aggregate set of locations are far apart from each other. A large diameter ratio may also reflect the case where the locations in one prefix are tightly clustered but the locations in the second are not. A large diameter ratio implies that aggregating the contiguous prefixes would remove the ability to express geographic routing policies.

Figure 2(b) shows the *diameter ratio* for each pair of contiguous prefixes in the routing table. We were surprised to see that smaller contiguous prefixes (*i.e.*, those in the /24-/31 range) spanned a greater geographic distance than larger contiguous prefixes (this phenomenon is shown in both Figure 2(a) and 2(b)). This geographic diversity is reflected along all three metrics (*i.e.*, number of distinct locations, maximum distance between IP addresses, and diameter ratio). Upon further examination, we found that this phenomenon can be explained by the fact that many ISPs based in the United States receive large prefix allocations and divide the allocation along /24 boundaries, advertising different /24s from different cities. On the other hand, we observe that ISPs in Europe and Asia typically advertise prefixes that correspond more closely with their actual allocations, which are usually considerably larger than /24. For example, in Europe, AS 5089 (NTL Group Limited, UK) advertises two separate contiguous /15s—80.2.0.0/15 and 80.4.0.0/15—for hosts in Cambridge and Luton, which are only about 75 miles apart.

To understand the extent to which the CIDR Report could be overestimating the opportunities for aggregation, we performed a CIDR Report style calculation on our dataset too. The CIDR Report computes the reduction in the number of contiguous prefixes when contiguous prefixes with same origin AS and AS path are aggregated. A similar calculation on our *Breadth* dataset showed that the number of prefixes advertised can be reduced by 64% if we aggregate. However, aggregating geographically diverse prefixes could conflict with the traffic engineering goals of an AS. Hence, if we aggregate only the prefixes that in addition to having similar AS paths, are geographically "close" (we used diameter ratio ≤ 500 as a definition for "close"), then the number of announced prefixes could be reduced by only 20%. Thus, the CIDR Report could be overestimating the opportunities for aggregation by a factor of 3.

[4]When a set of contiguous prefixes had different mask lengths, we classified the prefixes according to the *minimum* mask length in the set.

5. Conclusion

This paper studied the geographic properties of IP prefixes and their implications on Internet routing. Our findings have important implications not only for network applications that use IP prefixes to cluster end hosts, but also for Internet addressing. Advertising routes on a granularity that more closely reflects geographic locations (whether by renumbering, or by changing the addressing scheme entirely) could reduce routing table size by creating opportunities for aggregation.

Acknowledgments

We thank Neil Spring for providing access to and support for `undns`. This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the NSF under Cooperative Agreement No. ANI-0225660. Michael Freedman is supported by an NDSEG Fellowship. Nick Feamster is partially supported by an NSF Graduate Research Fellowship.

REFERENCES

[1] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. In *IEEE Global Internet Symposium*, Nov 2002.
[2] CIDR report. `http://www.cidr-report.org/`, 2005.
[3] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *NSDI*, Mar 2004.
[4] L. Gao. On inferring automonous system relationships in the Internet. *IEEE/ACM Trans. on Networking*, 9(6):733–745, Dec 2001.
[5] G. Huston. Growth of the BGP table, 1994 to present. `http://bgp.potaroo.net/`, 2005.
[6] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *ACM SIGCOMM*, Aug 2000.
[7] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. IPv4 address allocation and the BGP routing table evolution. *ACM SIGCOMM CCR*, 35(1):71–80, 2005.
[8] D. Meyer. University of Oregon RouteViews Project. `http://www.routeviews.org/`, 2005.
[9] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *ACM SIGCOMM*, Aug 2001.
[10] PlanetLab. `http://www.planet-lab.org/`, 2005.
[11] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *ACM SIGCOMM*, Aug 2003.

An Information-theoretic Approach to Network Monitoring and Measurement

Yong Liu
ECE Dept.
Polytechnic University
Brooklyn, NY 11201

Don Towsley
Dept. of Computer Science
University of Massachussetts
Amherst, MA 01003

Tao Ye
Sprint ATL
Burlingame, CA 94010

Jean Bolot
Sprint ATL
Burlingame, CA 94010

Abstract

Network engineers and operators are faced with a number of challenges that arise in the context of network monitoring and measurement. These include: i) how much information is included in measurement traces and by how much can we compress those traces?, ii) how much information is captured by different monitoring paradigms and tools ranging from full packet header captures to flow-level captures (such as with NetFlow) to packet and byte counts (such as with SNMP)? and iii) how much joint information is included in traces collected at different points and can we take advantage of this joint information? In this paper we develop a network model and an information theoretic framework within which to address these questions. We use the model and the framework to first determine the benefits of compressing traces captured at a single monitoring point, and we outline approaches to achieve those benefits. We next consider the benefits of joint coding, or equivalently of joint compression of traces captured a different monitoring points. Finally, we examine the difference in information content when measurements are made at either the flow level or the packet/byte count level. In all of these cases, the effect of temporal and spatial correlation on the answers to the above questions is examined. Both our model and its predictions are validated against measurements taken from a large operational network.

1 Introduction

Network monitoring is an immense undertaking in a large network. It consists of monitoring (or sensing) a network using a geographically distributed set of monitoring stations, with the purpose of using the monitoring data to better understand the behavior of the network and its users. Monitoring is a central activity in the design, engineering, and operation of a network. Increased monitoring capabilities, along with the associated increased understanding of the network and user behavior (or misbehavior), have direct impact on network performance and integrity, and therefore on the costs passed on to network users, and on the revenues of the operators. In spite of its importance, practitioners struggle with challenging, yet very practical questions such as where within the network to monitor data and at what granularity to capture traces, how much information is included in various types of packet traces and by how much can we compress those traces, and how much joint information is included in traces collected at different points and how can we take advantage of this joint information?

In this paper, we address these questions in the context of large high speed networks such as campus, enterprise, or core networks. Monitoring the behavior of such networks raises tremendous challenges due to the high bandwidth of currently deployed links. For example, the collection of 60-byte packet headers can easily generate 3Tb of data per hour on a OC-192 link (10 Gb/s link) in a core backbone, and 30Gb of data per hour at an enterprise or campus gateway. One means for reducing the amount of data gathered is to monitor flow-level data, as is done with Net Flow [2]. The amount of data can be further reduced by monitoring packet or byte counts over fixed intervals of time as is possible using SNMP [15]. Network data collected at distributed monitors also exhibit spatial and temporal correlations. Thus another means for reducing the sizes of monitored data sets is to exploit this correlation through correlated data coding and compression.

In this paper, we propose an *information theoretic framework* within which to address some of the issues and questions introduced above. In particular, we propose and validate a flow-level model (adapted from [12]), and we use it to determine the information content of a packet trace collected at a single or multiple points in a network, and of sets of packet traces collected at separate points. We also determine the information content of traces captured at different levels of granularity, in particular flow level traces (NetFlow traces) and byte or packet count traces (SNMP traces).

We obtain a number of interesting and important results. Regarding traces collected at a single monitoring point, we derive an information theoretic bound for the information content in those traces, or equivalently for the potential benefit of lossless compression on those traces. Not surprisingly, we find that the information content is small in SNMP traces, higher in NetFLow traces, and extremely high in full packet level traces. More interestingly, we show that full packet header traces can be compressed in practice down to a minimum of 20% of their original size, and that the amount of compression is a function of the average flow size traversing that node: the larger the average flow size, the smaller the compression ratio.

Regarding traces collected at multiple monitoring points, we find that joint coding or compression of traces further reduces the marginally compressed traces at the individual monitors. Specifically, the joint compression ratio (or equivalently, the additional compression benefit brought by joint coding of traces) is low for SNMP or byte/packet count traces, higher for NetFlow or flow-level traces, and higher still for packet-level traces. This means, for example, that joint coding is not really useful for SNMP data collected from different monitoring points and sent back to a NOC or central analysis station. Since SNMP data reporting takes little bandwidth anyway, it makes sense not to invest in sophisticated coding techniques in this case. However, NetFlow or packet header capture and reporting can require a significant amount of bandwidth and storage. Our results show that, in this case, joint coding techniques have the potential to significantly reduce those bandwidth and storage requirements.

Information-theoretic concepts and approaches have been used in the past to examine a wide variety of networking issues such as connectivity [18] and traffic matrix estimation [26], anomaly detection [16], and compact traffic descriptors [8, 24] for network dimensioning and QoS monitoring. However, to our knowledge, our attempt is the first to introduce a framework within which to address all the questions of interest here, namely trace coding, correlated and joint coding, and trace content at multiple time scales.

There has been work on trace compression, however it has been heuristic in nature. For example, work described in [14, 23] proposed heuristics based on storing and compressing packet headers collected at a single monitor along with timing information in the form of flow records. Trajectory sampling exhibits some elements in common with distributed compression of monitored data [7]. Also related is work in the area of inverting sampled data, e.g., [9, 11]. Indeed, sampling can be thought of as a form of lossy compression and these papers are concerned with decoding the resulting traces. There is also an extensive body of work produced within the sensor networking and distributed signal processing communities; see [4, 6] and their references for examples. However, much of it is in the context of abstract models of how data is produced, e.g., Gaussian random fields, and is not directly applicable in the domain of network monitoring.

The rest of the paper is structured in three parts. In Section 2 we present the various elements of our framework, including relevant concepts from information theory, our network flow-level model, and the network traces that will be used throughout the paper. In Section 3, we describe the first application of our framework, specifically we derive the information content of packet header traces collected at a single monitoring station and examine the benefits of trace compression in this case. In Section 4, we examine the more general problem of correlated, or joint, coding of trace captured at several monitoring stations. Section 5 then applies the model to determine the loss in information content when either flow-level (Net Flow) or byte count (SNMP) summary is applied to the full trace. Section 6 concludes the paper.

2 Methodologies

In this section we provide the foundation for the rest of the paper. We begin by reviewing several concepts needed from information theory (Section 2.1) that form the basis for exploring information content in network traces. This is followed with a description of the network flow model to which they will be applied (Section 2.2). The section ends with a review of a collection of network traces used to validate and parameterize the model (Section 2.3).

2.1 Some concepts from Information Theory

We begin by introducing the concepts of entropy and entropy rate and their relation to data compression [5].

Definition 1 Shannon entropy. *Let X be a discrete random variable that takes values from χ. Let $p(x) = P(X = x)$, $x \in \chi$. The entropy of X is defined by*

$$H(X) = -\sum_{x\in\chi} p(x)\log_2 p(x)$$

Examples of X in our context would be the flow size measured in packets, the flow identifier, and the packet/byte count in a one second interval.

Now consider a stochastic process $X = \{X_n\}_{n=1}^{\infty}$ where X_n is discrete valued.

Definition 2 ***Entropy Rate.*** *The entropy rate of a discrete valued stochastic process X is defined by*

$$H(X) = \lim_{n\to\infty} \frac{H(X_1, X_2, \ldots, X_n)}{n}$$

when the limit exists.

The entropy rate represents the information rate conveyed by the stochastic process X. It provides an achievable lower bound on the number of bits per sample required for lossless compression of the process. With lossless compression, all of the information is completely restored when the file is uncompressed. In our context, an example might be the byte counts at a link over successive one second intervals.

Definition 3 Joint Entropy Rate. *The joint entropy rate of a collection of many stochastic processes $\{X_n^{(i)}\}_{n=1}^{\infty}, i = 1, 2, ..., N$ is defined by*

$$\begin{aligned} &H(X^{(1)}, X^{(2)}, \ldots, X^{(N)}) \\ &= \lim_{n\to\infty} \frac{H((X_1^{(1)}, \ldots, X_n^{(1)}), \ldots, (X_1^{(N)}, \ldots, X_n^{(N)}))}{n} \end{aligned} \quad (1)$$

when the limit exists.

The joint entropy rate represents the information rate conveyed by the joint stochastic process and is an achievable lower bound on the number of bits required per sample for the joint lossless compression of all the processes.

Let us place this in our context. Let X_i be the header of the i-th packet and M the size of the header. $\{X_i\}_{i=1}^{\infty}$ is a stochastic process representing packet headers. We are interested in quantifying the benefit gained from compressing a packet header trace gathered from one network monitor or traces collected at a set of network monitors.

Definition 4 ***Marginal Compression Ratio.*** *Given stationary stochastic process $\{X_i\}_{i=1}^{\infty}$, the marginal compression ratio is defined as the ratio of the entropy rate and record size,*

$$\rho_m(X) = \frac{H(X)}{M}$$

In the case of traces collected at multiple points within the network, we define:

Definition 5 ***Joint Compression Ratio.*** *Given a collection of N jointly stationary stochastic processes $\{X_i^{(n)}\}_{i=1}^{\infty}, i = 1, 2, \ldots, N$, the joint compression ratio is defined as the ratio of the joint entropy rate and the sum of the entropy rates of the individual processes.*

$$\rho_j(X^{(1)}, X^{(2)}, \ldots, X^{(N)}) = \frac{H(X^{(1)}, X^{(2)}, \ldots, X^{(N)})}{\sum_{i=1}^{N} H(X^{(i)})}.$$

In the context of network trace compression, the joint compression ratio quantifies the potential benefits of jointly compressing the traces collected at several point in the network beyond simply compressing each trace independent of each other.

Although much of the time we will deal with discrete random variables, some quantities, such as interarrival times, are best approximated by continuous random variables. This necessitates the following definition.

Definition 6 ***Differential Entropy.*** *Let X be a continuous random variable with a density $f(X)$. The differential entropy of X is defined by*

$$h(X) = -\int_S f(x) \log f(x) dx$$

where S is the support set of the random variable.

In reality, every variable is measured with finite resolution. With a resolution of $\epsilon = 2^{-n}$, i.e., an n-bit quantization, a continuous random variable X is represented by a discrete random variable X^{ϵ} and its entropy is approximately $h(X) + n$.

If X follows an exponential distribution with rate λ, its differential entropy

$$h(X) = -\int_2^{\infty} \lambda e^{-\lambda x} \log_2(\lambda e^{-\lambda x}) dx = \log_2 \frac{e}{\lambda}$$

With an n-bit quantization, the discrete entropy of X is $H(X^{\epsilon}) = \log_2 \frac{e}{\lambda} + n$. In the following, whenever there is no confusion, we use the notation $H(X)$ for a continuous random variable X to represent its discrete entropy $H(X^{\epsilon})$.

2.2 Network Flow Model

In this section we introduce a *flow-based* network model. We represent the network as a directed graph $G = (V, E)$. Assume that flows arrive to the network according to a Poisson process with rate Λ and Δ_j denotes the inter-arrival time between flow j and $j-1$. Let $\Theta_j \in \mathcal{F}$ be the id of the j-th flow that arrives to the network. We assume that the route of a flow $f \in \mathcal{F}$ is fixed and classify flows into non-overlap flow classes, $\{\mathcal{F}_i, 1 \le i \le N\}$, such that all flows within a class share the same route in the monitored network. The route taken by flows in class i is represented by an ordered set

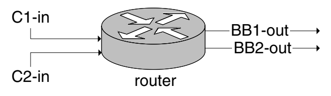


Figure 1: Experiment setup.

$R^{\langle i\rangle} = (v_1^{\langle i\rangle}, v_2^{\langle i\rangle}, \ldots, v_{l^{\langle i\rangle}}^{\langle i\rangle})$, where $v_j^{\langle i\rangle}$ is the j-th router traversed by a class i flow and $l^{\langle i\rangle}$ is the path length. The flow arrival rate within class i is $\Lambda^{\langle i\rangle} = \Lambda \times P(\Theta \in \mathcal{F}_i)$. When flow j arrives, it generates K_j packets. Packets within flow j arrive according to some point process with inter-arrival times $\{\delta_{j,i}\}_{i=2}^{K_j}$, where $\delta_{j,i}$ is the inter-arrival time between the $i-1$th and ith packet of flow j. It is assumed that the first packet arrives at the same time as the flow. The behavior of packet arrivals in the network is described by the stochastic process $\{(\Delta_j, \Theta_j, K_j, \{\delta_{j,i}\}_{i=2}^{K_j})\}$.

In practice, network traces are collected at distributed network monitors. We are interested in how information is carried around by packets when they traverse distributed network monitors. As a starting point, we assume that there is no packet loss in the network and packets incur constant delay on each link: let $D_{(i,j),k}$ denote the delay that the k-th packet incurs while traversing the k-th link, $(i,j) \in E$, we assume that $D_{(i,j),k} = D_{(i,j)}$, $\forall k$. Delays are very small and losses non-existent in a well-provisioned network such as the Sprint network. Hence this is a reasonable assumption in many cases. In Section 4.3 we describe how these assumptions can be relaxed. For a node v in the network, let $C^{(v)} \subseteq F$ denote the set of flows that pass through it. Since flows arrive to the network according to a Poisson process and the delay between any two nodes in the network is constant, flows arrive to node v according to a Poisson process with rate $\Lambda^{(v)} = \Lambda \times P(\Theta \in C^{(v)})$. The behavior of packet arrivals at node v can be described by the stochastic process $\{(\Delta_j^{(v)}, \Theta_j^{(v)}, K_j^{(v)}, \{\delta_{j,i}^{(v)}\}_{i=2}^{K_j^{(v)}})\}$, where $\{\Delta_j^{(v)}\}$ is the sequence of inter-flow-arrival times at node v that follows exponential distribution with rate $\Lambda^{(v)}$, $\{\Theta_j^{(v)}\}$ is an i.i.d. sequence of flow ids seen by v, $\{K_j^{(v)}\}$ is an i.i.d, sequence of integer valued random variables that denote the number of packets in the jth flow passing through v and $\{\delta_{j,i}^{(v)}\}_{i=2}^{K_j^{(v)}}$ is the sequence of flow inter packet arrival times.

Figure 1 illustrates a simple scenario corresponding to a router with two incoming links and two outgoing links, each of which contains a monitor. There are four two hop paths and four flow classes traversing these paths. This is the setting within which the traces used for validation and parameterization of the model are collected. They are described in the next section.

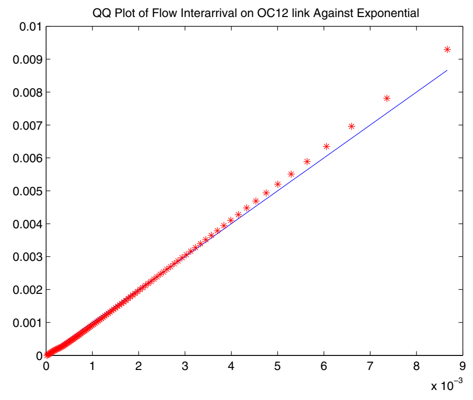


Figure 2: Flow arrival is Poisson

2.3 Validation

2.3.1 Network Traces

The data used here were collected as part of the full router experiment [13] on August 13, 2003. Packet traces were recorded at six interfaces, i.e. 12 links of one gateway router of the Sprint IP backbone network for 13 hours. We use mutually synchronized DAG cards [1] to record packet headers with GPS-precisioned 64bit timestamps from OC-3, OC-12 and OC-48 links [10]. The experiment setup is shown in Figure 1. In the remainder of this paper, we will focus on the traces collected over a one hour time period between 14:30 and 15:30 GMT on two incoming and two outgoing links. Table 1 summarizes the size of these traces and the link utilizations.

We choose this set of traces because it captures a mix of customer to gateway and gateway to backbone traffic. Customer traffic largely goes to one of the two backbone links. It is an ideal first step to analyze information redundancy without being overwhelmed by complicated routing. Furthermore, it is a representative set of traffic data, because there are over 500 similarly configured gateway routers in Sprint's global network. The shortcoming is that we do not have any backbone to backbone traffic recorded at the same time. This will be considered in the future work.

We further select the three most highly utilized links from the set to use in the remainder of the paper. BB1-out, BB2-out are from two OC-48 linecards connecting to two backbone routers (BB1 and BB2). C1-in and C2-in are from two OC-3 linecards connecting to transpacific customers. We use the full packet trace to deduce an SNMP-like utilization trace (refered to as utilization trace from now on) and an unsampled raw Netflow trace. This is done so that different network measurement techniques can be compared for the same time duration without any possible measurement errors con-

Data Set	duration	#packets	Average rate
C1-in	1 hour	44,598,254	53Mbps
C2-in	1 hour	123,542,790	73Mbps
BB1-out	1 hour	71,115,632	69Mbps
BB2-out	1 hour	105,868,070	82Mbps

Table 1: Trace description and stats.

tributed by SNMP or Netflow. The utilization trace is computed as byte-count per second rather than per 5 minutes as a normal SNMP trace. This is done to increase the number of data points and minimize estimation error in the calculation. We simulate Netflow by creating an unsampled Netflow trace from the packet trace, because we prefer to not take into account packet sampling's effect on measurement at this stage.

The full packet traces described above are used to validate the Poisson assumptions made in the flow model. Figure 2 depicts a QQ plot of the empirical inter-flow arrival time distribution with respect to an exponential distribution with the same average. We observe visually a good match for the flow inter-arrival times associated with link C1-in. This is also the case with the other flow arrival traces and is consistent with observations made elsewhere [12]. We compute empirical entropy on the traces described above using empirical marginal probability distributions.

3 Application: Single Point Trace Compression

All packet monitoring poses tremendous challenges to the storage subsystems due to the high volume of current network links. We use the information theoretic approach to identify and quantify the potential benefit of network trace compression based on the network flow model. In this section, we focus on full trace collection at a *single* monitor such as an IPMON system [10]. We start with calculating the information content in traces made of packet headers. It will become clear that the *temporal* correlation resulted from the flow structure leads to considerable marginal compression gain. We then show by empirical study that other fields in IP header will contain no/very small amount of information. We conclude this section with guidelines for the development of practical network trace compression algorithms. The next section concerns the simultaneous collection of traces at multiple monitors distributed throughout a network.

3.1 Entropy of Flow-based Trace

In the IPMON style full packet header trace capture system, The IP header and additional TCP/UDP header of each packet is stored in the trace. Here we only concern with the information content in the IP header, and leave TCP/UDP header for future work (except the port fields). We start with calculating the information content in packet time-stamp and 5-tuple flow ID.

The total length of an uncompressed timestamp is 64. The behavior of packet arrivals in the network is described by the stochastic process $\{(\Delta_j, \Theta_j, K_j, \{\delta_{j,i}\}_{i=2}^{K_j})\}$ (to avoid triviality, we assume $P(K_j > 1) = 1$). We are interested in determining the minimum number of bits required to represent each flow. If we assume flow inter-arrival $\{\Delta_j\}$, flow Id $\{\Theta_j\}$ and packet inter-arrivals within a flow $\{\delta_{j,i}\}_{i=2}^{K_j}$ are pairwise independent, on average we need a number of bits per flow equal to $H(\Phi)$ where

$$H(\Phi) = H(\Theta) + H(\Delta) + H(\{\delta_i\}_{i=2}^{K}), \quad (2)$$

where $H(\{\delta_i\}_{i=2}^{K})$ denotes the information content in packet inter-arrivals within a flow. It can be shown that

$$\begin{aligned} H(\{\delta_i\}_{i=2}^{K}) &= H(K) + H(\{\delta_i\}_{i=2}^{K}|K) && (3) \\ &= H(K) + \sum_{k=2}^{\infty} P(K=k) H(\{\delta_i\}_{i=2}^{k}|K=k) && (4) \\ &\le H(K) + \sum_{k=2}^{\infty} P(K=k)(k-1) H(\delta_i|K=k) && (5) \\ &\le H(K) + (E[K]-1) H(\delta), && (6) \end{aligned}$$

where δ represents the inter-arrival time between randomly picked adjacent packet pairs from all flows. Inequality (5) is an equality if packet inter-arrival times within a flow, $\{\delta_i\}$, are i.i.d. sequence. Inequality (6) is an equality if packet inter-arrival times are independent of flow size K.

Fig. 3 shows there is in fact a strong correlation between flow size and inter-packet arrivals time within a flow. Large flows with many packets tend to have smaller inter-packet arrival times. This suggests there is opportunity in further compressing the inter-packet arrival time within a flow. However the Inequality (6) provides us with an upper bound in compression ratio.

The per-flow information consists of two parts: one part is timing information about the flow arrival and flow ID, which is shared by all packets in the flow; the other part consists of all the packet inter-arrival information, which grows sub-linearly with the number of packets within the flow if we assume packet inter-arrivals are dependent. (Note: If packet inter-arrival times are independent it grows linearly.) The information rate per unit time is then $\Lambda H(\Phi)$.

Trace	$H(\Delta)$ bin=8μs	$H(K)$ bin=2pkts	$H(\delta)$ bin=128μs	$H(\Theta)$	$E(K)$	$H(total)$	$\rho(\Phi)$	Compression Algorithm
C1-in	8.8071	3.2168	9.7121	104	21.0039	706.3772	**0.2002**	**0.6425**
BB1-out	7.6124	2.4688	12.1095	104	20.4825	736.1722	**0.2139**	**0.6574**
BB2-out	7.1594	2.7064	12.4824	104	18.7890	689.9066	**0.2186**	**0.6657**

Table 2: Comparision of entropy calculations and real compression algorithm gain

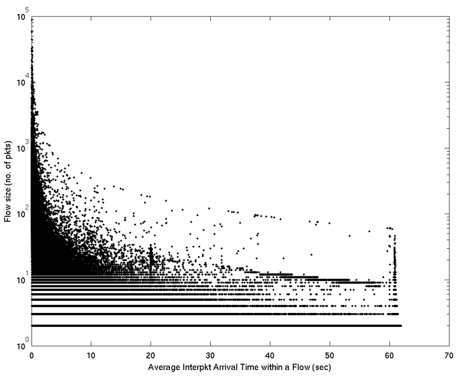


Figure 3: Flow size correlation with inter-packet arrival time within a flow. Larger flow size correspond to shorter inter-packet arrival times.

There exists other fields in the IP header as well, such as TOS, datagram size, etc. A detailed study in [19] shows that they carry little information content in the framework of the flow model, and can be modeled similarly to the packet inter-arrival time within a flow. We therefore take the simplified approach and only consider the timestamp field and the flow ID field in this paper.

3.2 Marginal Compression Ratio

In practice, traces are collected at individual nodes. For a node v in the network, we need a number of bits per flow equal to $H(\Phi^{(v)})$ where

$$H(\Phi^{(v)}) \leq H(\Theta^{(v)}) + H(\Delta^{(v)}) + H(K^{(v)}) + (E[K^{(v)}] - 1)H(\delta^{(v)}) \tag{7}$$

As before the inequality becomes equality when $\{K^{(v)}\}$ is independent of $\{\delta_{f,i}^{(v)}\}$ and the latter sequence is iid.

The information rate per unit time is then $\Lambda^{(v)}H(\Phi^{(v)})$ at node v. In the absence of compression, each flow requires on average $(104 + 64)E[K^{(v)}]$ bits with 104 bits to encode the 5-tuple flow identifier and 64 bits for timestamps of packet arrivals.

Now we can answer the question: what is the maximum benefit that can be achieved through compression? From $\Phi^{(v)}$, we have a *marginal compression ratio*

$$\rho(\Phi^{(v)}) = \frac{H(\Phi^{(v)})}{168 * E[K^{(v)}]} \tag{8}$$

The compression ratio $\rho(\Phi^{(v)})$ provides a lower bound on what can be achieved through lossless compression of the original network trace. From (7) and (8), the marginal compression ratio at a node is a decreasing function of $E[K^{(v)}]$, the average size of flows traversing that node. Since the information in flow ID $\Theta^{(v)}$ and flow arrival $\Delta^{(v)}$ is shared by all packets in the flow, *the larger the average flow size, the smaller the per-packet share, therefore the smaller the compression ratio*. When $E[K^{(v)}]$ is large, the compression ratio is bounded from below by $(E[H(\delta^{(v)})])/168$, which is an indication of how compressible the packet inter-arrival time is in average. (Note: this bound results from the assumption that packet inter-arrival times within a flow are independent. If there is correlation between packet inter-arrival times, a tighter bound can be derived to explore the correlation.) *Therefore, the marginal compression ratio for long flows is determined by the compression ratio of packet inter-arrival times.*

3.3 Results

We summarized the marginal compression ratio for a single trace in Table 2. We compare the compression upperbound from the flow model entropy calculation and a practical trace compression algorithm. The trace compression software is an implementation of the schemes introduced in [14]. The practical algorithm uses a flow-based compressed scheme very similar to the flow model in section 2.2. It does, however, record additional IPID, datagram size and TCP/UDP fields in each packet of a flow.

For the three traces we considered, the entropy calculation suggests a bound of compression ratio 20.02% to 21.86% of the original trace. However the practical scheme only compresses 64.25% to 66.57% of the original trace. The most significant algorithm difference is that this scheme keeps a fixed packet buffer for each flow, therefore records very long flows as many smaller flows. This is done for the ease of decompression and quickly count for very long flows, avoiding only outputting long flows after they end. This could result in long flows being broken down to many small flows and duplicate the flow record many times, therefore reducing the compression benefits.

4 Application: Joint Trace Coding

The fact that one flow traverses multiple monitors introduces *spatial* correlation among traces collected by distributed monitors. We calculate the joint entropy of network traces which serves as the lower bound for distributed traces compression. To account for network uncertainties, such as packet loss and random delay, we incorporate additional terms to characterize *network impairment*.

4.1 Joint Compression Ratio

We are interested in quantifying how well marginal compression comes to achieving the entropy rate of the network. We have

$$\rho_j(\Phi) = \frac{\Lambda H(\Phi)}{\sum_{v\in V} \Lambda^{(v)} H(\Phi^{(v)})} \tag{9}$$

where the numerator is the lower bound on joint compression and the denominator is the lower bound of marginal compression of each trace separately. The joint compression ratio ρ_j shows the benefit of joint compression.

The entropy of a flow record of class i can be calculated as $H(\Phi^{\langle i\rangle})$. The information rate generated by flow class i is $\Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle})$. Therefore, the network information rate is:

$$\Lambda H(\Phi) = \sum_{i=1}^{N} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle}) \tag{10}$$

At node v, denote by $D^{\langle v\rangle}$ the set of flow classes traversing it. The total information rate at v is

$$\Lambda^{(v)} H(\Phi^{(v)}) = \sum_{i\in D^{\langle v\rangle}} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle}) \tag{11}$$

Plugging (10) and (11) into (9), we have

$$\rho_j = \frac{\sum_{i=1}^{N} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle})}{\sum_{v\in V} \sum_{i\in D^{\langle v\rangle}} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle})} \tag{12}$$

$$= \frac{\sum_{i=1}^{N} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle})}{\sum_{i=1}^{N} l^{\langle i\rangle} \Lambda^{\langle i\rangle} H(\Phi^{\langle i\rangle})} \tag{13}$$

It suggests that the joint compression ratio is inversely proportional to a weighted average of the number of monitors traversed by flows. Intuitively, all monitors traversed by a flow collect redundant information for that flow. The longer the flows paths, the higher the spatial correlation in distributed packet header traces, the bigger the potential gain of distributed trace compression.

	Joint Compression Ratio
ρ(C1-in, BB1-out, C2-in, BB2-out)	**0.5**
ρ(C1-in to BB1-out)	**0.8649**
ρ(C1-in to BB2-out)	**0.8702**
ρ(C2-in to BB1-out)	**0.7125**
ρ(C2-in to BB2-out)	**0.6679**

Table 3: Joint trace compression ratio

4.2 Entropy Results

According to (9), the joint compression ratio for all the links traversing through our router is simply 0.5, since the path length (l) is 2 for each flow class. We're also interested in knowing the joint compression ratio of a path connecting any two links. For example, let us denote the flow classes C1-in to BB1-out as A, C1-in to BB2-out as B, and C2-in to BB1-out as C, the compression ratio for the path through links C1-in and BB1-out is:

$$\rho = \frac{H(BB1-out, C1-in)}{H(BB1-out) + H(C1-in)} \tag{14}$$

$$= \frac{\Lambda^A H(A) + \Lambda^B H(B) + \Lambda^C H(C)}{2\Lambda^A H(A) + \Lambda^B H(B) + \Lambda^C H(C)} \tag{15}$$

The results in Table 3 show a substantial 50% to 87% joint compression ratio over multiple monitoring points. The ratio is lower (namely 0.5) when computed across all monitors from the same router, and higher (namely 0.6679 to 0.8702) when computed over only two links. The savings come from a significant amount of shared flows in these monitoring points. This suggests that if we can successfully share information at the correlated monitoring points and perform joint coding, the potential savings will be very significant. In practice, it could be difficult to devise a perfect coding scheme for the entire network. We leave this for a future direction.

4.3 Network Impairments

Till now we have assumed that link delays are constant and that there are no packet losses. These can be accounted for by augmenting the expression for the flow entropy to account for these. Take the case where link delays are not constant as an example. Due to variable link delays, the inter-arrival time of two adjacent packets within a flow varies when they traverse the network. Link delays may also induce packet reordering. To fully capture the timing information of all packets in the network, one will have to record not only the packet inter-arrival time within a flow at their network entry point, but also the packet delays on all links. Suppose for now that the delay sequences $\{D_{(i,j),k}\}_{k=1}^{\infty}$, $(i,j) \in E$ are mutually independent sequences of iid rvs. Then $H(\Phi)$ becomes

$$H(\Phi) = H_w(\Phi) + H_n(\Phi)$$

where $H_w(\Phi)$ is given by (2) and $H_n(\Phi)$ is

$$H_n(\Phi) = E[K](H(D_{(i,j)}))$$

Note that in the case that link delays are correlated, this provides an upper bound. Figure 4 plots the distribution of the single hop packet delay on an operational router in Sprint Network. The delay is quantized with resolution of one micro-second, which corresponds to the accuracy of clock synchronization between two link monitors. From the figure, although the delay is not constant, it has skewed distribution. The empirical entropy is 4.2615, which means, in average, we need no more than 5 bits per packet to represent the variable packet delays across this router. Furthermore, we expect that packet delays are temporally correlated. By exploiting the temporal correlation, one can using even smaller number of bits to represent packet delays.

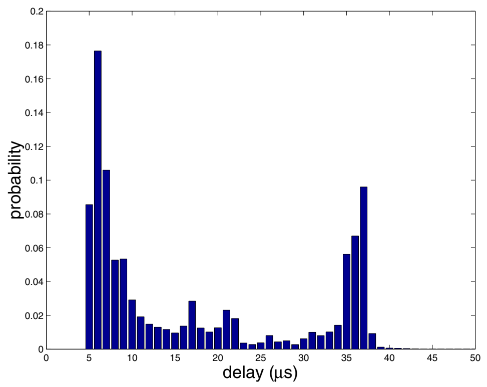


Figure 4: Single Hop Delay on Operational Router

The common belief is there is very little loss in large operational backbone networks, since they are usually over-provisioned. However, we expect that losses will introduce an additional term in the expression for $H_n()$. In particular, when losses are rare, they are likely to show up in the form of the entropy of a Bernoulli process, one for each link in the network. This will be the subject of future investigation.

Another impairment occurs when the routes change within the network. Fortunately, route changes are infrequent [25]. When they do occur, the changes must be recorded in the traces. However, they increase the information rate per unit time by an insignificant amount.

4.4 Development of Joint Compression Algorithms

Our models predict there is considerable gain to conduct joint packet header trace compression. It opens up the question of how one can develop algorithms to achieve the predicted joint compression ratio.

One direction is to develop a *centralized* algorithm to compress traces collected on distributed monitors. This requires all monitors to send their traces to a centralized server, which will generate a compressed aggregate trace. This approach will incur communication overhead in transmitting all those traces to the server. To reduce the communication overhead, traces can be aggregated on their ways to the server. Our spatial correlation model can be used to optimally organize the transmission and compression of traces. This is similar to the problem of joint routing and compression in sensor networks [22].

Alternatively, one can develop a *distributed* trace compression algorithm. Distributed data compression [5] aims at compressing correlated sources in a distributed way and achieving the gain of joint compression. How monitors compress packet traces without exchanging packet headers remains to be a challenging problem. It may also be possible to borrow ideas from *trajectory sampling* [7] to design joint compression algorithms.

5 Application: Information Content of Different Measurement Methods

In this section we apply information theory to evaluate the information gained from monitoring a network at different granularities. In addition to the packet header monitoring paradigm as exemplified by IPMON [14], we examine two other typical monitoring options. The most widely used monitoring method is SNMP utilization data collection [15], which is provided by all routers by default and has very low storage requirement. A second option is Cisco NetFlow [2] and its equivalent, which requires better router support and more demanding storages and analysis support. Clearly packet header monitoring is the most demanding and requires the greatest amount of resources,v however it provides the most comprehensive amount of information about the network. We first adopt the flow modelling of the network from section 2.2 to study both NetFlow and SNMP monitoring options. We then explore the quantitative difference among these methods using entropy calculation applied to the models.

5.1 A Model for NetFlow

Recording complete packet header traces is expensive and, thus, not commonly done. Cisco NetFlow is provided by some Cisco linecards to summarize flows without capturing every packet header. A raw NetFlow trace consists of a set of flow records. Each flow record includes the start time and duration of the flow, and the size of the flow (number of packets, bytes). In this section we explore the potential benefits of compression, both at a single monitor and across multiple monitors for this type of trace. In practice, NetFlow can sample packets on high volume links. For this paper, we consider the case where all packets are observed to derive flow records, hence avoiding inaccurate representation of flows from packet sampling described in [11] and [3].

A NetFlow record consists of an arrival time (flow inter-arrival time), flow ID, flow size, and the flow duration. Very similar to the flow model in Section 3.1, the average number of bit per flow record is

$$H(\Psi) = H(\Delta) + H(\Theta) + H(K)$$

We assume the five tuple flow ID is not compressed since it does not repeat in the capturing window.

In practice, each node individually turns on NetFlow and captures information for flows traversing it. The entropy of a NetFlow record at node v can be calculated as

$$H(\Psi^{(v)}) = H(\Delta^{(v)}) + H(\Theta^{(v)}) + H(K^{(v)}) \qquad (16)$$

Similar to Section 3.1, we can calculate the entropy of a NetFlow record of flow class i as $H(\Psi^{\langle i\rangle})$. The joint compression ratio of NetFlow traces can be calculated as

$$\rho_j(\Psi) = \frac{\Lambda H(\Psi)}{\sum_{v\in V} \Lambda^{(v)} H(\Psi^{(v)})} = \frac{\sum_{i=1}^{N} \Lambda^{\langle i\rangle} H(\Psi^{\langle i\rangle})}{\sum_{i=1}^{N} l^{\langle i\rangle} \Lambda^{\langle i\rangle} H(\Psi^{\langle i\rangle})}$$

The joint compression ratio is again inversely proportional to a weighted average of the number of monitors traversed by a flow. This suggests that NetFlow traces without sampling preserve the spatial correlation contained in full flow level packet trace. We show the joint compression ratio results based on our trace in Table 5.

So far, we assume NetFlow processes all the packets in all the flows. In a real network environment, both the number of flows and the number of packets generated by those flow are large. NetFlow can employ flow sampling, where only a fraction of flows are monitored, and packet sampling, where only a fraction of packets are counted, to reduce its cost in computation and memory, etc. Flow sampling will proportionally reduce the amount of flow information one obtained from the network. If two monitors sample flows independently with probability p, the chance that one flow gets sampled at both monitors is p^2, which leads to a reduced spatial correlation in NetFlow traces collected by these two monitors. Packet sampling will introduce errors in flow size and flow duration estimation. Furthermore, NetFlow records collected by two monitors for the same flow will be different. By aggregating distributed NetFlow traces, one can obtain more accurate flow information. We will extend our NetFlow model to account for flow sampling and packet sampling in future work.

5.2 A Model for SNMP

In this section we study the information content of SNMP measurements. By SNMP measurements, we mean packet/byte counts over fixed intervals of time. We begin with the flow model from Section 2.2. Let $\{(R_1(t), \ldots, R_{|E|}) : t \geq 0\}$, be the *rate process* associated with the network. In other words, $R_i(t)$ is the packet rate on link $i \in E$ at time $t \geq 0$. We will argue that it can be modeled as a multivariate Gaussian process and will calculate its associated parameters.

Let $\{R^j(t) : t \geq 0\}$ be the rate process associated with flow class $j \in [1, N]$, in other words, $R^j(t)$ is the rate at which flows within class $j \in [0, N]$ generate packets at time $t \geq 0$. Note that the processes associated with the different flow classes are independent. We have

$$R_i(t) = \sum_{j\in\mathcal{S}(i)} R^j(t), \quad i \in E \qquad (17)$$

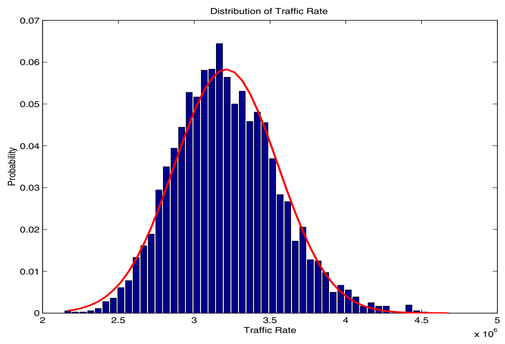
Here $\mathcal{S}(i) \subseteq [1, N]$ is the set of flow classes, whose flows pass through link $i \in E$. If we define the routing matrix $\{A_{ij}, 1 \leq i \leq |E|, 1 \leq j \leq M\}$ such that $A_{ij} = 1$ if link i is on the path of flow class j and $A_{ij} = 0$ otherwise, we have

$$R_I = AR^J, \qquad (18)$$

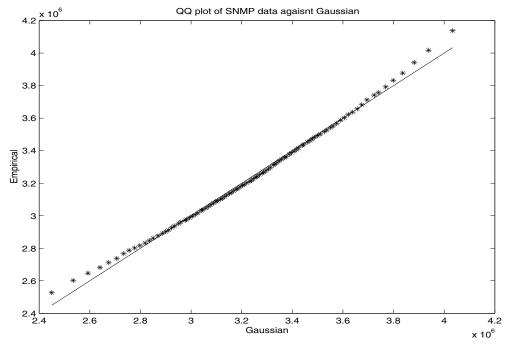
where $R_I = \{R_i, 1 \leq i \leq |E|\}$ is the rate vector on all network links and $R^J = \{R^j, 1 \leq j \leq N\}$ is the rate vector of all flow classes.

It has been observed that the traffic rate on high speed link tends to be Gaussian [17]. We further assume that the rate processes associated with the flow classes are independent Gaussian processes. Figure 5 plots the distribution of the traffic rate of one flow class in the trace under study. The Q-Q plot against Gaussian distribution shows a good match. The rate vector of all flow classes follows a multi-variate Gaussian distribution with mean $\mu^J = E[R^J] = \{\mu^j\}$ and covariance $K^J = cov(R^J) = \{K^J_{mn}\}$, where $K^J_{mn} = \sigma_j^2$ if $m = n = j$ and 0 otherwise. Consequently, the link rate vector is also a multi-variate Gaussian process with

$$\begin{aligned} \mu_I &= E[R_I] = A\mu^J && (19) \\ K_I &= cov(R_I) = AK^J A^T && (20) \end{aligned}$$



(a) Empirical Distribution



(b) QQ Plot Against Gaussian Distribution

Figure 5: Marginal Distribution of SNMP Data

5.2.1 Entropy in SNMP Data

According to [5], the marginal entropy of SNMP data collected on link i can be calculated as:

$$h(R_i) = \frac{\log 2\pi e \sum_{j=1}^{N} A_{ij}\sigma_j^2}{2} = \frac{\log 2\pi e \sum_{j\in\mathcal{S}(i)} \sigma_j^2}{2} \tag{21}$$

The joint entropy of SNMP data collected on link i and j can be calculated as:

$$\begin{aligned} h(R_i, R_l) &= \frac{\log\left\{(2\pi e)^2 \left| \begin{bmatrix} A_{i\cdot} \\ A_{l\cdot} \end{bmatrix} K^J \begin{bmatrix} A_{i\cdot}^T & A_{l\cdot}^T \end{bmatrix} \right| \right\}}{2} \\ &= \frac{\log\left\{(2\pi e)^2 (K_{11}K_{22} - K_{12}^2)\right\}}{2}, \end{aligned}$$

where $A_{i\cdot}$ is the i-th row vector in routing matrix A and $K_{11} = \sum_{j\in\mathcal{S}(i)} \sigma_j^2$, $K_{22} = \sum_{j\in\mathcal{S}(j)} \sigma_j^2$ and $K_{21} = \sum_{j\in\mathcal{S}(i)\cap\mathcal{S}(j)} \sigma_j^2$. Therefore

$$\begin{aligned} h(R_i, R_l) &= \frac{\log\left\{(2\pi e)^2 K_{11}K_{22}\right\}}{2} + \frac{\log\{1 - \frac{K_{12}^2}{K_{11}K_{22}}\}}{2} \\ &= h(R_i) + h(R_l) + \frac{\log(1-\rho_{il}^2)}{2}, \end{aligned}$$

where $\rho_{il} = \frac{K_{12}}{\sqrt{K_{11}K_{22}}}$ is the covariance coefficient between R_i and R_l. The mutual information between SNMP data on link i and link l is

$$\begin{aligned} I(R_i, R_l) &= h(R_i) + h(R_l) - h(R_i, R_l) & (22) \\ &= -\frac{\log(1-\rho_{il}^2)}{2}, & (23) \end{aligned}$$

which could be small even if ρ_{il} is close to one. For example, suppose there are 900 flows traversing both link i and link l. In addition to those common flows, links i and l each have their own 100 flows. If we assume all flows are statistically homogeneous, then we will have $\rho_{il} = 0.9$, which indicates R_i and R_l are highly correlated. However, according to (22), the mutual information between R_i and R_l is only around 1.2 bits. This suggests there is not much gain in doing joint compression of SNMP data.

5.2.2 How much information does SNMP contain about traffic matrices?

For the purpose of traffic engineering, it is important to characterize the traffic demand between all pairs of network ingress and egress points, or the Traffic Matrix (TM). Each element in the TM corresponds to the traffic rate of the flow class which goes from the ingress point to the egress point. Such information is normally not easy to access. Network operators can instrument each router to record SNMP data on each link, which is the sum of traffic rates of all flow classes traversing that link. It is challenging to infer the TM, equivalently the flow rate vector, based on the SNMP rate vector [26]. Our framework provides a way to quantify the amount of information about a TM that one can obtain from SNMP data.

Essentially, the information content in the flow rate vector (or TM) is

$$\begin{aligned} h(R^J) &= \sum_{j\in[0,N]} h(R^j) & (24) \\ &= \sum_{j\in[0,N]} \frac{1}{2}\log 2\pi e\sigma_j^2. & (25) \end{aligned}$$

Since the link rate vector is a linear combination of the flow rate vector, we will always have

$$h(R_I) \le h(R^J). \tag{26}$$

Furthermore, under the Gaussian assumption, the information content in the link rate vector can be calculated as

$$\begin{aligned} h(R_I) &= \frac{1}{2}\log\{(2\pi e)^{|E|}|K_I|\} && (27)\\ &= \frac{1}{2}\log\{(2\pi e)^{|E|}|AK^JA^T|\} && (28)\\ &\leq h(R^J) = \frac{1}{2}\log\{(2\pi e)^N|K^J|\} && (29) \end{aligned}$$

The gap in (29) is determined by both the routing matrix A and the variance in the traffic rates of flow classes $\{\sigma_j^2\}$.

Note: For some routing matrices, the rows of A are *dependent*, i.e., some link rate R_l is a linear combination of some other link rates. In this case, R_l contributes no new information to the link rate vector; consequently, we should only include all *independent* row vectors of A in (28) to calculate the entropy of the whole link rate vector. We will illustrate this through an example in Section5.3.

5.2.3 Entropy Rate in SNMP

In this section, we study the entropy rate in SNMP data by taking into account the temporal correlation in traffic rate processes. We have observed that the marginal link rate vector is well characterized as a multivariate Gaussian random variable. Consequently the link rate vector process over time is a stationary multivariate Gaussian process with mean μ_I given by (19) and covariance matrix at time lag h, $\Gamma_I(h)$, to be determined next. In order to simplify the discussion, we ignore link delays. These can be accounted for, but at the cost of some obfuscation. We have

$$\Gamma_I(h) = A \times \Gamma^J(h) \times A^T,$$

where $\Gamma^J_{mn}(h) = G^j(h)$ if $m = n = j$ and 0 otherwise, and $G^j(h)$ is the covariance associated with the class j rate process, $j \in [1, N]$. To obtain $\Gamma^J(h)$, we return to the original flow model and recognize that each flow class can be modeled as an M/G/∞ queue and that we can apply known results regarding the autocorrelation function of its buffer occupancy process, [21]. If T^j denotes the duration of a class f flow and $\hat{T}^j$ denotes the forward recurrence time associated with T^j, $j \in [1, N]$, then

$$G^j(h) = \Lambda^j E[T^j]\Pr[\hat{T}^j > |h|], \quad h \geq 0$$

The distribution of $\hat{T}^j$ is

$$\Pr[\hat{T}^j > h] = \frac{1}{E[T^j]}\int_h^\infty \Pr[T^j > x]dx, \quad h \geq 0$$

Suppose that we sample over intervals of length τ. Then we can use a discrete time version of the model where

$$R^j_k = \int_{(k-1)\tau}^{k\tau} R^j(t)dt, \quad 1 \leq j \leq N$$

Note that $\{R^j_k : k = 1, \ldots\}$ is a discrete time Gaussian process with mean $\mu^j\tau$ and covariance approximately equal to $\Gamma^\tau(h) = \Lambda^j E[T^j]\Pr[\hat{T}^j > |h\tau|]$, $h = 0, 1, 2, \ldots$. The discrete time counterparts for the rate process at the routers can be defined in a similar manner. The entropy rate of $\{R^j_k\}$ is given as, [5, Theorem 9.4.1]

$$\begin{aligned} h(\{R^j_k\}) &= \lim_{n\to\infty}\frac{1}{n}\frac{1}{2}\log(2\pi e)^n|K^j_n|,\\ &= \frac{1}{2}\log(2\pi e) + \lim_{n\to\infty}\frac{1}{2n}\log|K^j_n| \end{aligned}$$

where K^j_n is the $n \times n$ covariance matrix with elements $\Gamma^\tau(h)$, $h = 0, 1, \ldots, n-1$. Finally, the entropy rate of the system is $\sum_{j=1}^N h(\{R^j_k\})$.

5.2.4 SNMP Evaluation

The SNMP model is derived from the flow model and represents a summary of flow information. We evaluate the SNMP model by comparing the model derived entropy with an empirical entropy estimation.

The SNMP data is a per second utilization summarized from our trace data, instead of real per five minute data collected from the field. This is done to provide compatibility with the other monitoring methods. The empirical entropy estimation for each individual link is based on calculating empirical probability distribution function from the SNMP data, with a bin size of 50,000 bytes/sec. We use the BUB entropy estimator [20] with the PDF to derive the entropy. The BUB estimator does well when the number of available samples is small compared to the number of bins, as is the case here. It also provides an error bound for the estimation. For joint entropy such as H(BB1-out, C1-in), we compute the joint probability distribution of each SNMP data pair at time t: (BB1-out(t), C1-in(t)), then compute the entropy using the BUB function. In Table 4 we verify that the entropy of the SNMP data derived from the Gaussian model indeed matches well with the empirical calculations.

5.3 Results

We present results on the comparison of the three monitoring options in terms of quantitative storage difference and distributed compression savings. The joint compression ratio indicates whether two links share a strong spatial correlation. The stronger the links are correlated, the lower the compression ratio. This is intuitive because when two links share a large amount of information, the shared information only needs to be recorded once, hence yielding a large compression savings. In Table 5, we find that the spatial correlation is very weak at the SNMP level (compression ratio is approx. 1), but much

Data Set	Entropy	Model	Empirical
SNMP	H(BB1-out)	20.6862	20.6700 (max mse<0.1499)
	H(BB2-out)	20.8895	20.8785 (max mse<0.1654)
	H(C1-in)	21.0468	21.0329 (max mse<0.1618)
	H(C1-in,BB1-out)	26.1517	26.1254(max mse<0.2717)
	H(C1-in,BB2-out)	26.1432	26.3078(max mse<0.2945)

Table 4: Comparision of SNMP data entropy between model and empirical calculations

Data Set	Entropy (bits)	Joint Compression Ratio
SNMP	C1-in 21.0468	C1-in and BB1-out 1.0021
	BB1-out 20.6862	C1-in and BB2-out 0.9997
NetFlow	C1-in 160.8071*697	C1-in and BB1-out 0.8597
	BB1-out 159.6124*1730	C1-in and BB2-out 0.8782
Full Trace	C1-in 706.3773*697	C1-in and BB1-out 0.8649
	BB1-out 736.1722*1730	C1-in and BB2-out 0.8702

Table 5: Comparison of entropy calculations in information gain for different measurement granularity

stronger at both NetFlow and all packet monitoring levels (compression ratio is between 0.5 to 0.8702). This result suggests that there is no need to coordinate SNMP data gathering at different monitoring points, while coordinated collection and shared information at all monitoring points can yield significant savings in terms of storage for widely deployed NetFlow and all packet monitoring.

Table 5 also shows the information content comparison among the three monitoring options. SNMP data takes about 21bits to encode per second, while NetFlow takes 74444bits per second, and all packet monitoring takes 492082bits per second.

Now let's turn to the information gap between SNMP data and actual flow rate vector as studied in Section5.2.2. For the case under study, there are 4 links and 4 flow classes between 4 incoming-outgoing link pairs. The routing matrix is

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Row vectors of A are dependent:

$$A_{4\cdot} = A_{1\cdot} + A_{2\cdot} - A_{3\cdot},$$

which means you can obtain the rate on link 4 as

$$R_4 = R_1 + R_2 - R_3.$$

Therefore,

$$\begin{aligned} h(R_I) &= h(R_1, R_2, R_3) && (30) \\ &= \frac{1}{2}\log\{(2\pi e)^3 |A_{1-3,\cdot} K^J A^T_{1-3,\cdot}|\} && (31) \end{aligned}$$

We obtained the rate variance of each flow classes as:

$$K^J = \begin{bmatrix} 1.21 & 0 & 0 & 0 \\ 0 & 1.18 & 0 & 0 \\ 0 & 0 & 0.239 & 0 \\ 0 & 0 & 0 & 0.42 \end{bmatrix} \times 10^{11}$$

According to (29), the information content in the rate vector for all 4 flow classes is $h(R^J) = 79.87$; according to (28), the information content in the rate vector for all 4 links is $h(R_I) = 61.08$. The information gap is 23.53% of the total flow rate information. For this very simple topology and routing matrix, even if we collect SNMP data on all incoming and outgoing links, we still cannot fully infer the traffic rates between all incoming and outgoing pairs. We conjecture that as the topology and routing gets more complex, the information gap between the SNMP link rate vector and the flow rate vector increases, in other words, it becomes more difficult to obtain the traffic matrix by just looking at the SNMP data.

6 Conclusion and Future Work

Our goal in this paper was to put together a framework in which we could pose, and answer, challenging yet very practical questions faced by network researchers, designers, and operators, specifically i) how much information is included in various types of packet traces and by how much can we compress those traces, and ii) how much joint information is included in traces collected at different points and how can we take advantage of this joint information?

We obtained a number of interesting results. For example, we derived an information theoretic bound for the information content in traces collected at a single monitoring point and therefore were able to quantify the potential benefit of lossless compression on those traces. Not surprisingly, we found that the compression ratio (or information content) is small in SNMP traces, much higher in NetFLow traces, and extremely high in full packet traces. This shows that full packet capture does provide a quantum leap increase in information about network behavior. However, deploying full packet capture stations can be expensive. The interesting comparison, then, is that between the additional cost of full packet capture (say compared to NetFlow capture), and the additional amount of information produced by full packet traces (say compared to NetFlow traces). We are currently working on this problem, but early results seem to indicate that the increase in information content is proportionally larger than the increase in cost, suggesting that a full packet monitoring system gives you ”more bang for the buck” (or rather ”more entropy for the buck”).

We also found that full packet header traces can be compressed in practice down to a minimum of 20% of their original size, and that the amount of compression is a function of the average flow size traversing that node: the larger the average flow size, the smaller the compression ratio.

In practice, packet traces are typically captured at multiple points. Therefore, it is important to understand how much information content is available in a set of traces, when that set is considered as a whole (as opposed to as a set of independent traces). This is turn is crucial to tackle further problems such as how many monitoring stations to set up (there might be a point of diminishing returns at which additional stations do not bring in enough ”fresh” new information) and how to process and analyze those correlated data traces.

Using our framework, we find that joint coding or compression of traces further reduces the marginally compressed traces at the individual monitors. Specifically, the joint compression ratio (or equivalently, the additional compression benefit brought by joint coding of traces) is low for SNMP or byte/packet count traces, higher for NetFlow or flow-level traces, and significantly higher for packet-level traces. This means, for example, that joint coding would be very useful for full packet trace data (and to a lesser extent for NetFlow data) collected from different monitoring points and sent back to a NOC or central analysis station. We are now working on extending the work in this paper to joint coding techniques for large scale backbone and wireless networks.

References

[1] http://dag.cs.waikato.ac.nz/.

[2] http://www.cisco.com/warp/public/732/tech/netflow. *Cisco NetFlow*.

[3] Choi, B.-Y., and Bhattacharyya, S. On the accuracy and overhead of cisco sampled netflow. In *Sigmetrics Workshop on Large Scale Network Inference (LSNI): Methods, Validation, and Applications* (June 2005).

[4] Chou, J., Petrovic, D., and Ramchandran, K. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *IEEE Infocom 2003* (April 2003).

[5] Cover, T. A., and Thomas, J. A. *Information theory*. John Wiley & Sons, Inc., 1991.

[6] Cristescu, R., Beferull-Lozano, B., and Vetterli, M. On network correlated data gathering. In *INFOCOM* (Hong Kong, 2004).

[7] Duffield, N., and Grossglauser, M. Trajectory sampling with unreliable reporting. In *IEEE Infocom* (March 2004).

[8] Duffield, N., Lewis, J. T., O'Connell, N., Russell, R., and Toomey, F. Entropy of atm traffic streams: A tool for estimating qos parameters. *IEEE JSAC*, 13 (1995), 981–990.

[9] Duffield, N., Lund, C., and Thorup, M. Properties and prediction of flow statistics from sampled packet streams. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (New York, NY, USA, 2002), ACM Press, pp. 159–171.

[10] Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T., and Diot, C. Packet-level traffic measurements from the sprint IP backbone. *IEEE Network* (2003).

[11] Hohn, N., and Veitch, D. Inverting sampled traffic. In *ACM/SIGCOMM Internet Measurement Conference* (November 2003).

[12] Hohn, N., Veitch, D., and Abry, P. Cluster process, a natural language for network traffic. In *IEEE Transactions on Signal Processing, Special Issue on Signal Processing in Networking* (August 2003), vol. 51, pp. 2229–2244.

[13] Hohn, N., Veitch, D., Papagiannaki, K., and Diot, C. Bridging router performance and queuing theory. In *ACM Sigmetrics* (New York, 2004).

[14] IANNACCONE, G., DIOT, C., GRAHAM, I., AND MCKEOWN, N. Monitoring very high speed links. In *Proceedings of ACM Internet Measurement Workshop* (November 2001).

[15] K. MCCLOGHRIE, M. T. R. Rfc 1213.

[16] LEE, W., AND XIANG, D. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy* (2001).

[17] LELAND, W., TAQQU, M., WILLINGER, W., AND WILSON, D. V. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking 2*, 1 (February 1994).

[18] LIU, X., AND SRIKANT, R. An information-theoretic view of connectivity in wireless sensor networks. In *IEEE SECON (Conference on Sensor Networks and Adhoc Communications and Networks)* (2004).

[19] LIU, Y., TOWSLEY, D., WENG, J., AND GOECKEL, D. An information theoretic approach to network trace compression. Tech. Rep. UMASS CMPSCI 05-03, University of Massachusetts, Amherst, 2004.

[20] PANINSKI, L. Estimation of entropy and mutual information. *Neural Computation 15*, 1191-1254 (2003).

[21] PARULEKAR, M., AND MAKOWSKI, A. $M/G/\infty$ input processes: A versatile class of models for network traffic. In *Proceedings of INFOCOM* (1997).

[22] PATTEM, S., KRISHNAMACHARI, B., AND GOVINDAN, R. The impact of spatial correlation on routing with compression in wireless sensor networks. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks* (2004), pp. 28–35.

[23] PEUHKURI, M. A method to compress and anonymize packet traces. In *Proceedings of ACM Internet Measurement Workshop* (November 2001).

[24] PLOTKIN, N., AND ROCHE, C. The entropy of cell streams as a traffic descriptor in atm networks. In *IFIP Performance of Communication Networks* (October 1995).

[25] ZHANG, Y., DUFFIELD, N., PAXSON, V., AND SHENKER, S. On the constancy of internet path properties. In *Proc. ACM SIGCOMM Internet Measurement Workshop* (2001).

[26] ZHANG, Y., ROUGHAN, M., LUND, C., AND DONOHO, D. An information-theoretic approach to traffic matrix estimation. In *ACM SIGCOMM* (August 2003).

Multi-Hop Probing Asymptotics in Available Bandwidth Estimation: Stochastic Analysis

Xiliang Liu
City University of New York
xliu@gc.cuny.edu

Kaliappa Ravindran
City College of New York
ravi@cs.ccny.cuny.edu

Dmitri Loguinov
Texas A&M University
dmitri@cs.tamu.edu

Abstract

This paper analyzes the asymptotic behavior of packet-train probing over a multi-hop network path $\mathcal{P}$ carrying arbitrarily routed bursty cross-traffic flows. We examine the statistical mean of the packet-train output dispersions and its relationship to the input dispersion. We call this relationship the *response curve* of path $\mathcal{P}$. We show that the real response curve $\mathcal{Z}$ is *tightly* lower-bounded by its *multi-hop fluid counterpart* $\mathcal{F}$, obtained when every cross-traffic flow on $\mathcal{P}$ is hypothetically replaced with a constant-rate fluid flow of the same average intensity and routing pattern. The real curve $\mathcal{Z}$ asymptotically approaches its fluid counterpart $\mathcal{F}$ as probing packet size or packet train length increases. Most existing measurement techniques are based upon the single-hop fluid curve $\mathcal{S}$ associated with the bottleneck link in $\mathcal{P}$. We note that the curve $\mathcal{S}$ coincides with $\mathcal{F}$ in a certain large-dispersion input range, but falls below $\mathcal{F}$ in the remaining small-dispersion input ranges. As an implication of these findings, we show that bursty cross-traffic in multi-hop paths causes negative bias (asymptotic underestimation) to most existing techniques. This bias can be mitigated by reducing the deviation of $\mathcal{Z}$ from $\mathcal{S}$ using large packet size or long packet-trains. However, the bias is not completely removable for the techniques that use the portion of $\mathcal{S}$ that falls below $\mathcal{F}$.

1 Introduction

End-to-end estimation of the spare capacity along a network path using packet-train probing has recently become an important Internet measurement research area. Several measurement techniques such as TOPP [14], Pathload [6], IGI/PTR [5], Pathchirp [16], and Spruce [17] have been developed. Most of the current proposals use a single-hop path with constant-rate fluid cross-traffic to justify their methods. The behavior and performance of these techniques in a multi-hop path with general bursty cross-traffic is limited to experimental evaluations. Recent work [9] initiated the effort of developing an analytical foundation for bandwidth measurement techniques. Such a foundation is important in that it helps achieve a clear understanding of both the validity and the inadequacy of current techniques and provides a guideline to improve them. However, the analysis in [9] is restricted to single-hop paths. There is still a void to fill in understanding packet-train bandwidth estimation over a multi-hop network path.

Recall that the available bandwidth of a network hop is its residual capacity after transmitting cross-traffic within a certain time interval. This metric varies over time as well as a wide range of observation time intervals. However, in this paper, we explicitly target the measurement of a *long-term average* available bandwidth, which is a stable metric independent of observation time instances and observation time intervals [9]. Consider an N-hop network path $\mathcal{P} = (L_1, L_2, \ldots, L_N)$, where the capacity of link L_i is denoted by C_i and the long-term average of the cross-traffic arrival rate at L_i is given by λ_i, which is assumed to be less than C_i. The hop available bandwidth of L_i is $A_i = C_i - \lambda_i$. The path available bandwidth $A_{\mathcal{P}}$ is given by

$$A_{\mathcal{P}} = \min_{1 \le i \le N} (C_i - \lambda_i). \tag{1}$$

The hop L_b, which carries the minimum available bandwidth, is called the *tight* link or the bottleneck link[1]. That is,

$$b = \arg \min_{1 \le i \le N} (C_i - \lambda_i). \tag{2}$$

The main idea of packet-train bandwidth estimation is to infer $A_{\mathcal{P}}$ from the relationship between the inter-packet dispersions of the output packet-trains and those of the input packet-trains. Due to the complexity of this relationship in arbitrary network paths with bursty cross-traffic flows, previous work simplifies the analysis using a single-hop path with fluid[2] cross-traffic, while making the following two assumptions without formal justification: first, cross-traffic burstiness only causes measurement variability that can be smoothed out by averaging multiple probing samples and second, non-bottleneck links have negligible impact on the proposed techniques.

The validity of the first assumption is partially addressed in [9], where the authors use a single-hop path with bursty cross-traffic to derive the statistical mean of the packet-train output dispersions as a function of the input probing dispersion, referred to as the single-hop response curve. Their analysis shows that besides measurement variability, cross-traffic burstiness can also cause *measurement bias* to the techniques that are based on fluid analysis. This measurement bias *cannot* be reduced even when an infinite number of probing samples are used, but can be mitigated using long packet-trains and/or large probing packet size.

This paper addresses further the two assumptions that current techniques are based on. To this end, we extend the asymptotic analysis in [9] to arbitrary network paths and uncover the nature of the measurement bias caused by bursty cross-traffic flows in a *multi-hop* network path. This problem is significantly different from previous single-hop analysis due to the following reasons. First, unlike single-hop measurements, where the input packet-trains have deterministic and equal inter-packet separation formed by the probing source, the input packet-trains at any hop (except the first one) along a multi-link path are output from the previous hop and have random structure. Second and more importantly, the multi-hop probing asymptotics are strongly related to the routing pattern of cross-traffic flows. This issue never arises in a single-hop path and it has received little attention in prior investigation. However, as we show in this paper, it is one of the most significant factors that affect the accuracy of bandwidth measurement in multi-hop paths.

To characterize packet-train bandwidth estimation in its most general settings, we derive the probing response curve $\mathcal{Z}$ of a multi-hop path $\mathcal{P}$ assuming arbitrarily routed bursty cross-traffic flows. We compare $\mathcal{Z}$ with its multi-hop fluid counterpart $\mathcal{F}$, which is a response curve obtained when every cross-traffic flow in $\mathcal{P}$ is hypothetically replaced with a fluid flow of the same average intensity and routing pattern. We show, under an ergodic stationarity assumption for each cross-traffic flow, that the real curve $\mathcal{Z}$ is tightly lower bounded by its fluid counterpart $\mathcal{F}$ and that the curve $\mathcal{Z}$ asymptotically approaches its fluid bound $\mathcal{F}$ in the entire input range as probing packet size or packet-train length increases.

Most of the existing techniques are based on the single-hop fluid response curve $\mathcal{S}$ associated with the bottleneck link in $\mathcal{P}$. Therefore, any deviation of the real curve $\mathcal{Z}$ from the single-hop curve $\mathcal{S}$ can potentially cause measurement bias in bandwidth estimation. Note that the deviation $\mathcal{Z}-\mathcal{S}$ can be decomposed as

$$\mathcal{Z}-\mathcal{S}=(\mathcal{Z}-\mathcal{F})+(\mathcal{F}-\mathcal{S}). \qquad (3)$$

The first term $\mathcal{Z}-\mathcal{F}$ is always positive and causes asymptotic underestimation of $A_{\mathcal{P}}$ for most of the existing techniques. This deviation term and its resulting measurement bias are "elastic" in the sense that they can be reduced to a negligible level using packet-trains of sufficient length[3]. For the second deviation term $\mathcal{F}-\mathcal{S}$, we note that both $\mathcal{S}$ and $\mathcal{F}$ are piece-wise linear curves. The first two linear segments in $\mathcal{F}$ associated with large input dispersions coincide with $\mathcal{S}$ (i.e., $\mathcal{F}-\mathcal{S}=0$). The rest of the linear segments in $\mathcal{F}$ associated with small input dispersions appear above $\mathcal{S}$ (i.e., $\mathcal{F}-\mathcal{S}>0$). The amount of deviation and the additional negative measurement bias it causes are dependent on the routing patterns of cross-traffic flows, and are maximized when every flow traverses only one hop along the path (which is often called *one-hop persistent* cross-traffic routing [4]). Furthermore, the curve deviation $\mathcal{F}-\mathcal{S}$ is "non-elastic" and stays constant with respect to probing packet size and packet-train length at any given input rate. Therefore, the measurement bias it causes cannot be overcome by adjusting the input packet-train parameters.

Among current measurement techniques, pathload and PTR operate in the input probing range where $\mathcal{F}$ coincides with $\mathcal{S}$, and consequently are only subject to the measurement bias caused by the first deviation term $\mathcal{Z}-\mathcal{F}$. Spruce may use the probing range where $\mathcal{F}-\mathcal{S}>0$. Hence it is subject to both elastic and non-elastic negative measurement biases. The amount of bias can be substantially more than the actual available bandwidth in certain common scenarios, leading to negative results by the measurement algorithm and a final estimate of zero by the tool.

The rest of the paper is organized as follows. Section 2 derives the multi-hop response curve $\mathcal{F}$ assuming arbitrarily routed fluid cross-traffic flows and examines the deviation term $\mathcal{F}-\mathcal{S}$. In Section 3 and 4, we derive the real response curve $\mathcal{Z}$ of a multi-hop path and show its relationship to its fluid counterpart $\mathcal{F}$. We provide practical evidence for our theoretical results using testbed experiments and real Internet measurements in Section 5. We examine the impact of these results on existing techniques in Section 6 and summarize related work in Section 7. Finally, we briefly discuss future work and conclude in Section 8.

Due to limited space, most of the proofs in this paper are omitted, and we refer interested readers to [10] for more technical details.

2 Multi-Hop Fluid Analysis

It is important to first thoroughly understand the response curve $\mathcal{F}$ of a network path carrying fluid cross-traffic flows, since as we show later, the fluid curve $\mathcal{F}$ is an *approachable* bound of the real response curve $\mathcal{Z}$. Initial investigation of the fluid curves is due to Melandar *et al.* [13] and Dovrolis *et al.* [3]. However, prior work only considers two special cross-traffic routing cases (one-hop persistent routing and path persistent routing). In this section, we formulate and solve the problem for arbitrary cross-traffic routing patterns, based on which, we discuss several important properties of the fluid response curves that allow us to obtain the path available bandwidth information.

2.1 Formulating A Multi-Hop Path

We first introduce necessary notations to formulate a multi-hop path and the cross-traffic flows that traverse along the path.

An N-hop network path $\mathcal{P} = (L_1, L_2, \ldots, L_N)$ is a sequence of N interconnected *First-Come First-Served (FCFS) store-and-forward* hops. For each forwarding hop L_i in $\mathcal{P}$, we denote its link capacity by C_i, and assume that it has infinite buffer space and a work-conserving queuing discipline. Suppose that there are M fluid cross-traffic flows traversing path $\mathcal{P}$. The rate of flow j is denoted by x_j and the flow rate vector is given by $\mathbf{x} = (x_1, x_2, \ldots, x_M)$.

We impose two routing constraints on cross-traffic flows to simplify the discussion. The first constraint requires every flow to have a different routing pattern. In the case of otherwise, the flows with the same routing pattern should be aggregated into one single flow. The second routing constraint requires every flow to have only one link where it enters the path and also have only one (downstream) link where it exits from the path. In the case of otherwise, the flow is decomposed into several separate flows that meet this routing constraint.

Definition 1 *A flow aggregation is a set of flows, represented by a "selection vector"* $\mathbf{p} = (p_1, p_2, \ldots, p_M)^T$*, where* $p_j = 1$ *if flow* j *belongs to the aggregation and* $p_j = 0$ *if otherwise. We use* $\mathbf{f}_j$ *to represent the selection vector of the aggregation that contains flow* j *alone.*

There are several operations between flow aggregations. First, the common flows to aggregations $\mathbf{p}$ and $\mathbf{q}$ form another aggregation, whose selection vector is given by $\mathbf{p} \odot \mathbf{q}$, where the operator $\odot$ represents "element-wise multiplication." Second, the aggregation that contains the flows in $\mathbf{p}$ but not in $\mathbf{q}$ is given by $\mathbf{p} - \mathbf{p} \odot \mathbf{q}$. Finally, note that the traffic intensity of aggregation $\mathbf{p}$ can be computed from the inner product $\mathbf{xp}$.

We now define several types of flow aggregation frequently used in this paper. First, the traversing flow aggregation at link L_i, denoted by its selection vector $\mathbf{r}_i$, includes all fluid flows that pass through L_i. The $M \times N$ matrix $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ becomes the routing matrix of path $\mathcal{P}$. For convenience, we define an auxiliary selection vector $\mathbf{r}_0 = \mathbf{0}$.

The second type of flow aggregation, denoted by $\mathbf{e}_i$, includes all flows entering the path at link L_i, which can be expressed as $\mathbf{e}_i = \mathbf{r}_i - \mathbf{r}_i \odot \mathbf{r}_{i-1}$ given the second routing constraint stated previously. The third type of flow aggregation, which includes flows that enter the path at link L_k and traverse the downstream link L_i, is denoted as $\Gamma_{k,i} = \mathbf{e}_k \odot \mathbf{r}_i$, where $k \leq i$.

The cross-traffic intensity at link L_i is denoted by λ_i. We assume $\lambda_i < C_i$ for $1 \leq i \leq N$. Since none of the links in $\mathcal{P}$ is congested, the arrival rate of flow j at any link it traverses is x_j. Consequently, we have

$$\lambda_i = \mathbf{x}\mathbf{r}_i < C_i, \quad 1 \leq i \leq N. \tag{4}$$

We further define the *path configuration* of $\mathcal{P}$ as the following $2 \times N$ matrix

$$\mathbf{H} = \begin{pmatrix} C_1 & C_2 & \ldots & C_N \\ \lambda_1 & \lambda_2 & \ldots & \lambda_N \end{pmatrix}. \tag{5}$$

The hop available bandwidth of L_i is given by $A_i = C_i - \lambda_i$. We assume that every hop has different available bandwidth, and consequently that the tight link is unique. Sometimes, we also need to refer to the second minimum hop available bandwidth and the associated link, which we denote as $A_{b2} = C_{b2} - \lambda_{b2}$ and L_{b2}, respectively. That is

$$b2 = \arg \min_{1 \leq i \leq N, i \neq b} (C_i - \lambda_i), \tag{6}$$

where b is the index of the tight hop.

2.2 Fluid Response Curves

We now consider a packet-train of input dispersion (i.e., inter-packet spacing) g_I and packet size s that is used to probe path $\mathcal{P}$. We are interested in computing the output dispersion of the packet train and examining its relation to g_I. Such a relation is called the *gap response curve* of path $\mathcal{P}$. It is easy to verify that under fluid conditions, the response curve does not depend on the packet-train length n. Hence, we only consider the case of packet-pair probing. We denote the output dispersion at link L_i as $\gamma_i(g_I, s)$ or γ_i for short, and again for notational convenience we let $\gamma_0 = g_I$. Note that $\gamma_N(g_I, s)$ corresponds to the notation $\mathcal{F}$ we have used previously.

Based on our formulations, the gap response curve of path $\mathcal{P}$ has a recursive representation given below.

Theorem 1 *When a packet-pair with input dispersion* g_I *and packet size* s *is used to probe an* N*-hop fluid path with routing matrix* $\mathbf{R}$ *and flow rate vector* $\mathbf{x}$*, the output dispersion at link* L_i *can be recursively expressed as*

$$\gamma_i = \begin{cases} g_I & i = 0 \\ \max\left(\gamma_{i-1}, \dfrac{s + \Omega_i}{C_i}\right) & i > 0 \end{cases}, \tag{7}$$

where Ω_i *is* [4]

$$\Omega_i = \sum_{k=1}^{i} \Big[\gamma_{k-1}\mathbf{x}\Gamma_{k,i}\Big]. \tag{8}$$

Proof: Assumes that the first probing packet arrives at link L_i at time instance a_1. It gets immediate transmission service and departs at a_1+s/C_i. The second packet arrives at $a_1+\gamma_{i-1}$. The server of L_i needs to transmit $s+\Omega_i$ amount of data before it can serve the second packet. If this is done before time instance $a_1+\gamma_{i-1}$, the second packet also gets immediate service and $\gamma_i=\gamma_{i-1}$. Otherwise, the sever undergoes a busy period between the departure of the two packets, meaning that $\gamma_i=(s+\Omega_i)/C_i$. Therefore, we have

$$\gamma_i = \max\left(\gamma_{i-1}, \frac{s+\Omega_i}{C_i}\right). \tag{9}$$

This completes the proof of the theorem. ■

As a quick sanity check, we verify the compatibility between Theorem 1 and the special one-hop persistent routing case, where every flow that enters the path at link L_i will exit the path at link L_{i+1}. For this routing pattern, we have

$$\Gamma_{k,i} = \begin{cases} \mathbf{0} & i \neq k \\ \mathbf{r}_i & i = k \end{cases}. \tag{10}$$

Therefore, equation (8) can be simplified as

$$\Omega_i = \gamma_{i-1}\mathbf{x}\mathbf{r}_i = \gamma_{i-1}\lambda_i, \tag{11}$$

which agrees with previous results [3], [13].

2.3 Properties of Fluid Response Curves

Theorem 1 leads to several important properties of the fluid response curve $\mathcal{F}$, which we discuss next. These properties tell us how bandwidth information can be extracted from the curve $\mathcal{F}$, and also show the deviation of $\mathcal{F}$, as one should be aware of, from the single-hop fluid curve $\mathcal{S}$ of the tight link.

Property 1 *The output dispersion $\gamma_N(g_I,s)$ is a continuous piece-wise linear function of the input dispersion g_I in the input dispersion range $(0,\infty)$.*

Let $0=\alpha_{K+1}<\alpha_K<\ldots<\alpha_1<\alpha_0=\infty$ be the input dispersion turning points that split the gap response curve to $K+1$ linear segments[5]. Our next result discusses the turning points and linear segments that are of major importance in bandwidth estimation.

Property 2 *The first turning point α_1 corresponds to the path available bandwidth in the sense that $A_{\mathcal{P}}=s/\alpha_1$. The first linear segment in the input dispersion range $(\alpha_1=s/A_{\mathcal{P}},\infty)$ has slope 1 and intercept 0. The second linear segment in the input dispersion range (α_2,α_1) has slope λ_b/C_b and intercept s/C_b, where b is the index of the tight link:*

$$\gamma_N(g_I,s) = \begin{cases} g_I & \alpha_1 \le g_I \le \infty \\ \dfrac{g_I\lambda_b+s}{C_b} & \alpha_2 \le g_I \le \alpha_1 \end{cases}. \tag{12}$$

These facts are irrespective of the routing matrix.

It helps to find the expression for the turning point α_2, so that we can identify the exact range for the second linear segment. However, unlike α_1, the turning point α_2 is dependent on the routing matrix. In fact, all other turning points are dependent on the routing matrix and can not be computed based on the path configuration matrix alone. Therefore, we only provide a bound for α_2.

Property 3 *For any routing matrix, the term s/α_2 is no less than A_{b2}, which is the second minimum hop available bandwidth of path $\mathcal{P}$.*

The slopes and intercepts for all but the first two linear segments are related to the routing matrix. We skip the derivation of their expressions, but instead provide both a lower bound and an upper bound for the entire response curve.

Property 4 *For a given path configuration matrix, the gap response curve associated with any routing matrix is lower bounded by the single-hop gap response curve of the tight link*

$$\mathcal{S}(g_I,s) = \begin{cases} g_I & g_I > \dfrac{s}{A_{\mathcal{P}}} \\ \dfrac{s+g_I\lambda_b}{C_b} & 0 < g_I < \dfrac{s}{A_{\mathcal{P}}} \end{cases}. \tag{13}$$

It is upper bounded by the gap response curve associated with one-hop persistent routing.

We now make several observations regarding the deviation of $\gamma_N(g_I,s)$ (i.e., $\mathcal{F}$) from $\mathcal{S}(g_I,s)$. Combing (12) and (13), we see that $\gamma_N(g_I,s)-\mathcal{S}(g_I,s)=0$ when $g_I\ge\alpha_2$. That is, the first two linear segments on $\mathcal{F}$ coincide with $\mathcal{S}$. When $g_I<\alpha_2$, Property 4 implies that the deviation $\gamma_N(g_I,s)-\mathcal{S}(g_I,s)$ is positive. The exact value depends on cross-traffic routing and it is maximized in one-hop persistent routing for any given path configuration matrix.

Also note that there are three pieces of path information that we can extract from the gap response curve $\mathcal{F}$ without knowing the routing matrix. By locating the first turning point α_1, we can compute the path available bandwidth. From the second linear segment, we can obtain the tight link capacity and cross-traffic intensity (and consequently, the bottleneck link utilization) information. Other parts of the response curve $\mathcal{F}$ are less readily usable due to their dependence on cross-traffic routing.

2.4 Rate Response Curves

To extract bandwidth information from the output dispersion γ_N, it is often more helpful to look at the *rate* response curve, i.e., the functional relation between the output rate $r_O=s/\gamma_N$ and the input rate $r_I=s/g_I$. However, since this relation is not linear, we adopt a transformed version first proposed by Melander *et al.* [14], which depicts the relation between the ratio r_I/r_O and r_I. Denoting this rate response curve by $\tilde{\mathcal{F}}(r_I)$, we have

$$\tilde{\mathcal{F}}(r_I) = \frac{r_I}{r_O} = \frac{\gamma_N(g_I, s)}{g_I}. \tag{14}$$

This transformed version of the rate response curve is also piece-wise linear. It is easy to see that the first turning point in the rate curve is $s/\alpha_1 = A_p$ and that the rate curve in the input rate range $(0, s/\alpha_2)$ can be expressed as

$$\tilde{\mathcal{F}}(r_I) = \begin{cases} 1 & r_I \le A_{\mathcal{P}} \\ \dfrac{\lambda_b + r_I}{C_b} & \dfrac{s}{\alpha_2} \ge r_I \ge A_{\mathcal{P}} \end{cases}. \tag{15}$$

Finally, it is also important to notice that the rate response curve $\tilde{\mathcal{F}}(r_I)$ does not depend on the probing packet size s. This is because, for any given input rate r_I, both $\gamma_N(g_I, s)$ and g_I are proportional to s. Consequently, the ratio between these two terms remains a constant for any s.

2.5 Examples

We use a simple example to illustrate the properties of the fluid response curves. Suppose that we have a 3-hop path with equal capacity C_i = 10mb/s, $i = 1, 2, 3$. We consider two routing matrices and flow rate settings that lead to the same link load at each hop.

In the first setting, the flow rate vector $\mathbf{x} = (4, 7, 8)$ and the routing pattern is *one-hop* persistent, i.e., $\mathbf{R} = \text{diag}(1, 1, 1)$. In the second setting, the flow rate vector $\mathbf{x} = (4, 3, 1)$ and the routing pattern is *path* persistent. That is,

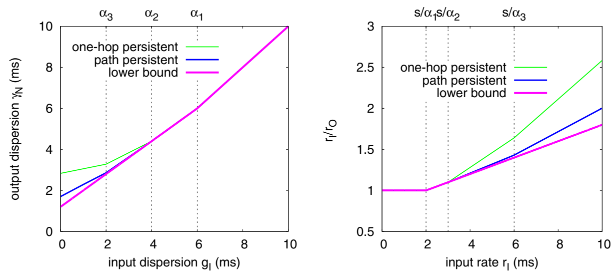
$$\mathbf{R} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}. \tag{16}$$

Both of the settings result in the same path configuration matrix

$$\mathbf{H} = \begin{pmatrix} 10 & 10 & 10 \\ 4 & 7 & 8 \end{pmatrix}. \tag{17}$$

The probing packet size s is 1500 bytes. The fluid gap response curves for the two routing patterns are plotted in Fig. 1(a). In this example, both curves have 4 linear segments separated by turning points α_1 = 6ms, α_2 = 4ms, and α_3 = 2ms. Note that part of the curve for path-persistent routing appears below the one for one-hop persistent routing. The lower bound $\mathcal{S}$ identified in Property 4 is also plotted in the figure. This lower bound is the gap response curve of the single-hop path comprising only the tight link L_3.

The rate response curves for the two examples are given in Fig. 1(b), where the three turning points are 2mb/s, 3mb/s, and 6mb/s respectively. Due to the transformation



(a) gap response curve (b) rate response curve

Figure 1: An example of multi-hop response curves.

we adopted, the rate curve for one-hop persistent routing still remains as an upper bound for the rate curves associated with the other routing patterns. From Fig. 1(b), we also see that, similar to the gap curves, the two multi-hop rate response curves and their lower bound $\tilde{\mathcal{S}}(r_I)$ (i.e., the transformed rate version of $\mathcal{S}(g_I, s)$) share the same first and second linear segments.

2.6 Discussion

We conclude this section by discussing several major challenges in extending the response curve analysis to a multi-hop path carrying bursty cross-traffic flows. First, notice that with bursty cross-traffic, even when the input dispersion and packet-train parameters remain constant, the output dispersion becomes random, rather than deterministic as in fluid cross-traffic. The gap response curve $\mathcal{Z}$, defined as the functional relation between the statistical mean of the output dispersion and the input dispersion, is much more difficult to penetrate than the fluid curve $\mathcal{F}$. Second, unlike in the fluid case, where both packet-train length n and probing packet size s have no impact on the rate response curve $\tilde{\mathcal{F}}(r_I)$, the response curves in bursty cross-traffic are strongly related to these two packet-train parameters. Finally, a full characterization of a fluid flow only requires one parameter – its arrival rate, while a full characterization of a bursty flow requires several stochastic processes. In what follows, we address these problems and extend our analysis to multi-hop paths with bursty cross-traffic.

3 Basics of Non-Fluid Analysis

In this section, we present a stochastic formulation of the multi-hop bandwidth measurement problem and derive a recursive expression for the output dispersion random variable. This expression is a fundamental result that the asymptotic analysis in Section 4 is based upon.

3.1 Formulating Bursty Flows

We keep most of the notations the same as in the previous section, although some of the terms are extended to have a different meaning, which we explain shortly. Since cross-traffic flows now become bursty flows of data packets, we adopt the definitions of several random processes (Definition 1-6) in [9] to characterize them. However, these definitions need to be refined to be specific to a given router and flow aggregation. In what follows, we only give the definitions of two random processes and skip the others. The notations for all six random processes are given in Table 3.1.

Definition 2 *The cumulative traffic arrival process of flow aggregation* $\mathbf{p}$ *at link* L_i*, denoted as* $\{V_i(\mathbf{p}, t), 0 \leq t < \infty\}$ *is a random process counting the total amount of data (in bits) received by hop* L_i *from flow aggregation* $\mathbf{p}$ *up to time instance* t*.*

Definition 3 *Hop workload process of* L_i *with respect to flow aggregation* $\mathbf{p}$*, denoted as* $\{W_i(\mathbf{p}, t), 0 \leq t < \infty\}$ *indicates the sum at time instance* t *of service times of all packets in the queue and the remaining service time of the packet in service, assuming that flow aggregation* $\mathbf{p}$ *is the only traffic passing through link* L_i*.*

We next make several modeling assumptions on cross-traffic flows. First, we assume that all flows have stationary arrivals.

Assumption 1 *For any cross-traffic flow* j *that enters the path from link* L_i*, the cumulative traffic arrival process* $\{V_i(\mathbf{f}_j, t)\}$ *has ergodic stationary increments. That is, for any* $\delta > 0$*, the* δ*-interval traffic intensity process* $\{Y_{i,\delta}(\mathbf{f}_j, t)\}$ *is a mean-square ergodic process with time-invariant distribution and ensemble mean* x_j*.*

We explain this assumption in more details. First, the stationary increment assumption implies that the increment process of $\{V_i(\mathbf{f}_j, t)\}$ for any given time interval δ, namely $\{V_i(\mathbf{f}_j, t + \delta) - V_i(\mathbf{f}_j, t) = \delta Y_{i,\delta}(\mathbf{f}_j, t)\}$, has a time-invariant distribution. This further implies that the δ-interval traffic intensity process $\{Y_{i,\delta}(\mathbf{f}_j, t)\}$ is *identically distributed*, whose marginal distribution at any time instance t can be described by the same random variable $Y_{i,\delta}(\mathbf{f}_j)$. Second, the mean-square ergodicity implies that, as the observation interval δ increases, the random variable $Y_{i,\delta}(\mathbf{f}_j)$ converges to x_j in the mean-square sense. In other words, the variance of $Y_{i,\delta}(\mathbf{f}_j)$ decays to 0 as $\delta \to \infty$, i.e.,

$$\lim_{\delta\to\infty} E\left[\left(Y_{i,\delta}(\mathbf{f}_j) - x_j\right)^2\right] = 0. \qquad (18)$$

Our next assumption states the independent relationship between different flows that enter path $\mathcal{P}$ at the same link.

$\{V_i(\mathbf{p}, t)\}$	Cumulative arrival process at L_i w.r.t. $\mathbf{p}$
$\{Y_{i,\delta}(\mathbf{p}, t)\}$	Cross-traffic intensity process at L_i w.r.t. $\mathbf{p}$
$\{W_i(\mathbf{p}, t)\}$	Hop workload process at L_i w.r.t. $\mathbf{p}$
$\{D_{i,\delta}(\mathbf{p}, t)\}$	Workload-difference process at L_i w.r.t. $\mathbf{p}$
$\{U_i(\mathbf{p}, t)\}$	Hop utilization process at L_i w.r.t. $\mathbf{p}$
$\{B_{i,\delta}(\mathbf{p}, t)\}$	Available bandwidth process at L_i w.r.t. $\mathbf{p}$

Table 1: Random process notations

Assumption 2 *For any two flows* j *and* l *that enter the path at link* L_i*, the two processes* $\{V_i(\mathbf{f}_j, t)\}$ *and* $\{V_i(\mathbf{f}_l, t)\}$ *are independent. Specifically, for any two time instances* t_1 *and* t_2*, the two random variables* $V_i(\mathbf{f}_j, t_1)$ *and* $V_i(\mathbf{f}_l, t_2)$ *are independent.*

As a consequence of the two assumptions we made, the ergodic stationary property also holds for any flow aggregations at their entering link.

Corollary 1 *For any flow aggregation* $\mathbf{p}$ *that enters the path at link* L_i*, i.e.,* $\mathbf{p} \odot \mathbf{e}_i = \mathbf{p}$*, the process* $\{V_i(\mathbf{p}, t)\}$ *has ergodic stationary increments. Consequently, the traffic intensity random variable* $Y_{i,\delta}(\mathbf{p})$ *converges to* $\mathbf{xp}$ *in the mean-square sense*

$$\lim_{\delta\to\infty} E\left[\left(Y_{i,\delta}(\mathbf{p}) - \mathbf{xp}\right)^2\right] = 0. \qquad (19)$$

Due to Szczotka [18], [19], the workload process $\{W_i(\mathbf{p}, t)\}$ will "inherit" the ergodic stationarity property from the traffic arrival process $\{V_i(\mathbf{p}, t)\}$. This property is further carried over to the δ-interval workload-difference process $\{D_{i,\delta}(\mathbf{p}, t)\}$ and the available bandwidth process $\{B_{i,\delta}(\mathbf{p}, t)\}$. This distributional stationarity allows us to focus on the corresponding random variables $W_i(\mathbf{p})$, $D_{i,\delta}(\mathbf{p})$, and $B_{i,\delta}(\mathbf{p})$. It is easy to get, from their definitions, that the statistical means of $D_{i,\delta}(\mathbf{p})$ and $B_{i,\delta}(\mathbf{p})$ are 0 and $C_i - \mathbf{xp}$, respectively[6]. Further, the ergodicity property leads to the following result.

Lemma 1 *For any flow aggregation* $\mathbf{p}$ *that enter the path at link* L_i*, the random variable* $B_{i,\delta}(\mathbf{p})$ *converges in the mean-square sense to* $C_i - \mathbf{xp}$ *as* $\delta \to \infty$*, i.e.,*

$$\lim_{\delta\to\infty} E\left[\left(B_{i,\delta}(\mathbf{p}) - (C_i - \mathbf{xp})\right)^2\right] = 0. \qquad (20)$$

On the other hand, notice that unlike $\{Y_{i,\delta}(\mathbf{p}, t)\}$ and $\{B_{i,\delta}(\mathbf{p}, t)\}$, the workload-difference process $\{D_{i,\delta}(\mathbf{p}, t)\}$ is not a moving average process by nature. Consequently, the mean-square ergodicity of $\{D_{i,\delta}(\mathbf{p}, t)\}$ does not cause the variance of $D_{i,\delta}(\mathbf{p})$ to decay with respect to the increase of δ. Instead, we have the following lemma.

Lemma 2 *The variance of the random variable* $D_{i,\delta}(\mathbf{p})$ *converges to* $2Var[W_i(\mathbf{p})]$ *as* δ *increases:*

$$\lim_{\delta\to\infty} E\left[\left(D_{i,\delta}(\mathbf{p}) - 0\right)^2\right] = 2Var\left[W_i(\mathbf{p})\right]. \qquad (21)$$

To obtain our later results, not only do we need to know the asymptotic variance of $Y_{i,\delta}(\mathbf{p})$, $D_{i,\delta}(\mathbf{p})$ and $B_{i,\delta}(\mathbf{p})$ when δ approaches infinity, but also we often rely on their variance being uniformly bounded (for any δ) by some constant. This condition can be easily justified from a practical standpoint. First note that cross-traffic arrival rate is bounded by the capacities of incoming links at a given router. Suppose that the sum of all incoming link capacities at hop L_i is C_+, then $Y_{i,\delta}(\mathbf{p})$ is distributed in a finite interval $[0, C_+]$ and its variance is uniformly bounded by the constant C_+^2 for any observation interval δ. Similarly, the variance of $B_{i,\delta}(\mathbf{p})$ is uniformly bounded by the constant C_i^2. The variance of $D_{i,\delta}(\mathbf{p})$ is uniformly bounded by the constant $4Var[W_i(\mathbf{p})]$ for any δ, which directly follows from the definition of $D_{i,\delta}(\mathbf{p})$.

Finally, we remind that some of the notations introduced in Section 2.1 now are used with a different meaning. The rate of the bursty cross-traffic flow j, denoted by x_j, is the probabilistic mean of the traffic intensity random variable $Y_{i,\delta}(\mathbf{f}_j)$, which is also the *long-term average* arrival rate of flow j at any link it traverses. The term $\lambda_i = \mathbf{x}\mathbf{r}_i$ becomes the long-term average arrival rate of the aggregated cross-traffic at link L_i. The term $A_i = C_i - \lambda_i$ is the long-term average hop available bandwidth at link L_i. Again recall that we explicitly target the measurement of long-term averages of available bandwidth and/or cross-traffic intensity, instead of the corresponding metrics in a certain time interval.

3.2 Formulating Packet Train Probing

We now consider an infinite series of packet-trains with input inter-packet dispersion g_I, packet size s, and packet-train length n. This series is driven to path $\mathcal{P}$ by a point process $\Lambda(t) = \max\{m \geq 0 : T_m \leq t\}$ with sufficient large inter-probing separation. Let $d_1(m,i)$ and $d_n(m,i)$ be the departure time instances from link L_i of the first and last probing packets in the m^{th} packet-train. We define the *sampling interval* of the packet-train as the total spacing $\Delta = d_n(m,i) - d_1(m,i)$, and the *output dispersion* as the average spacing $G = \Delta/(n-1)$ of the packet-train. Both Δ and G are random variables, whose statistics might depend on several factors such as the input dispersion g_I, the packet-train parameters s and n, the packet-train index m in the probing series, and the hop L_i that the output dispersion G is associated with. Therefore, a full version of G is written as $G_i(g_I, s, n, m)$. However, for notation brevity, we often omit the parameters that have little relevance to the topic under discussion.

We now formally state the questions we address in this paper. Note that a realization of the stochastic process $\{G_N(g_I, s, n, m), 1 \leq m < \infty\}$ is just a packet-train probing experiment. We examine the sample-path time-average of this process and its relationship to g_I when keeping s and n constant. This relationship, previously denoted by $\mathcal{Z}$, is called the gap response curve of path $\mathcal{P}$.

Notice that the ergodic stationarity of cross-traffic arrival, as we assumed previously, can reduce our response curve analysis to the investigation of a single random variable. This is because each packet-train comes to see a multi-hop system of the same stochastic nature and the output dispersion process $\{G_N(m), 1 \leq m < \infty\}$ is an *identically distributed* random sequence, which can be described by the output dispersion random variable G_N. The sample-path time average of the output dispersion process coincides with the mean of the random variable G_N[7]. Therefore, in the rest of the paper, we focus on the statistics of G_N and drop the index m.

In our later analysis, we compare the gap response curve of $\mathcal{P}$ with that of the *fluid counterpart* of $\mathcal{P}$ and prove that the former is lower-bounded by the latter.

Definition 4 *Suppose that path $\mathcal{P}$ has a routing matrix $\mathbf{R}$ and a flow rate vector $\mathbf{x}$ and that path $\tilde{\mathcal{P}}$ has a routing matrix $\tilde{\mathbf{R}}$ and a flow rate vector $\tilde{\mathbf{x}}$. $\tilde{\mathcal{P}}$ is called the fluid counterpart of $\mathcal{P}$ if 1) all cross-traffic flows traversing $\tilde{\mathcal{P}}$ are constant-rate fluid; 2) the two paths $\tilde{\mathcal{P}}$ and $\mathcal{P}$ have the same configuration matrix; and 3) there exists a row-exchange matrix T, such that $T\mathbf{R} = \tilde{\mathbf{R}}$ and $T\mathbf{x} = \tilde{\mathbf{x}}$.*

From this definition, we see that for every flow j in $\mathcal{P}$, there is a corresponding fluid flow j' in the fluid counterpart of $\mathcal{P}$ such that flow j' have the same average intensity and routing pattern as those of flow j. Note that the third condition in Definition 4 is made to allow the two flows have different indices, i.e., to allow $j \neq j'$.

A second focus of this paper is to study the impact of packet-train parameters s and n on the response curves. That is, for any given input rate r_I and other parameters fixed, we examine the convergence properties of the output dispersion random variable $G_N(s/r_I, s, n)$ as s or n tends to infinity.

3.3 Recursive Expression of G_N

We keep input packet-train parameters g_I, s, and n constant and next obtain a basic expression for the output dispersion random variable G_N.

Lemma 3 *Letting $G_0 = g_I$, the random variable G_i has the following recursive expression*

$$\begin{aligned} G_i &= \sum_{k=1}^{i} \frac{Y_{k,\Delta_{k-1}}(\Gamma_{k,i})G_{k-1}}{C_i} + \frac{s}{C_i} + \frac{\tilde{I}_i}{n-1} \\ &= G_{i-1} + \frac{D_{i,\Delta_{i-1}}(\mathbf{e}_i)}{n-1} + \frac{R_i}{n-1}, \end{aligned} \qquad (22)$$

where the term R_i is a random variable representing the extra queuing delay[8] *(besides the queuing delay caused by* *the workload process $\{W_i(\mathbf{e}_i, t)\}$) experienced at L_i by the last probing packet in the train. The term $\tilde{I}_i$ is another random variable indicating the hop idle time of L_i during the sampling interval of the packet train.*

This result is very similar to Lemma 5 in [9]. However, due to the random input packet-train structure at L_i, all but the term s/C_i in (22) become random variables. Some terms, such as $D_{i,\Delta_{i-1}}(\mathbf{e}_i)$ and $Y_{k,\Delta_{k-1}}(\Gamma_{k,i})$, even have two dimensions of randomness. To understand the behavior of probing response curves, we need to investigate the statistical properties of each term in (22).

4 Response Curves in Bursty Cross-Traffic

In this section, we first show that the gap response curve $\mathcal{Z} = E[G_N(g_I, s, n)]$ of a multi-hop path $\mathcal{P}$ is lower bounded by its fluid counterpart $\mathcal{F} = \gamma_N(g_I, s)$. We then investigate the impact of packet-train parameters on $\mathcal{Z}$.

4.1 Relation Between $\mathcal{Z}$ and $\mathcal{F}$

Our next lemma shows that passing through a link can only increase the dispersion random variable in mean.

Lemma 4 *For $1 \leq i \leq N$, the output dispersion random variable G_i has a mean no less than that of G_{i-1}. That is, $E[G_i] \geq E[G_{i-1}]$.*

Using the first part of (22), our next lemma shows that for any link L_i, the output dispersion random variable G_i is lower bounded in mean by a linear combination of the output dispersion random variables G_k, where $k < i$.

Lemma 5 *For $1 \leq i \leq N$, the output dispersion random variable G_i satisfies the following inequality*

$$E[G_i] \geq \frac{1}{C_i}\left(\sum_{k=1}^{i} \mathbf{x}\Gamma_{k,i} E[G_{k-1}] + s\right). \tag{23}$$

From Lemma 4 and Lemma 5, we get

$$E[G_i] \geq \max\left(E[G_{i-1}], \frac{\sum_{k=1}^{i} \mathbf{x}\Gamma_{k,i} E[G_{k-1}] + s}{C_i}\right). \tag{24}$$

This leads to the following theorem.

Theorem 2 *For any input dispersion g_I, packet-train parameters s and n, the output dispersion random variable G_N of path $\mathcal{P}$ is lower bounded in mean by the output dispersion $\gamma_N(g_I, s)$ of the fluid counterpart of $\mathcal{P}$:*

$$E[G_N(g_I, s, n)] \geq \gamma_N(g_I, s). \tag{25}$$

Proof: We apply mathematical induction to i. When $i = 0$, $E[G_0] = \gamma_0 = g_I$. Assuming that (25) holds for $0 \leq i < N$, we next prove that it also holds for $i = N$. Recalling (24), we have

$$\begin{aligned} E[G_N] &\geq \max\Big(E[G_{N-1}], \frac{\sum_{k=1}^{N} \mathbf{x}\Gamma_{k,N} E[G_{k-1}] + s}{C_N}\Big) \\ &\geq \max\Big(\gamma_{N-1}, \frac{\sum_{k=1}^{N} \mathbf{x}\Gamma_{k,N} \gamma_{k-1} + s}{C_N}\Big) = \gamma_N, \end{aligned}$$

where the second inequality is due to the induction hypothesis, and the last equality is because of Theorem 1. ■

Theorem 2 shows that in the entire input gap range, the piece-wise linear fluid gap response curve $\mathcal{F}$ discussed in Section 2 is a lower bound of the real gap curve $\mathcal{Z}$. The deviation between the real curve $\mathcal{Z}$ and its fluid lower bound $\mathcal{F}$, which is denoted by $\beta_N(g_I, s, n)$ or β_N for short, can be recursively expressed in the following, where we let $\beta_0 = 0$:

$$\beta_i = \begin{cases} \beta_{i-1} + \dfrac{E[R_i]}{n-1} & \gamma_i = \gamma_{i-1} \\ \dfrac{1}{C_i} \sum_{k=1}^{i} \mathbf{x}\Gamma_{k,i}\beta_{k-1} + \dfrac{E[\tilde{I}_i]}{n-1} & \gamma_i > \gamma_{i-1} \end{cases}. \tag{26}$$

In what follows, we study the asymptotics of the curve deviation β_N when input packet-train parameters s or n becomes large and show that the fluid lower bound $\mathcal{F}$ is in fact a *tight* bound of the real response curve $\mathcal{Z}$.

4.2 Impact of Packet Train Parameters

We now demonstrate that for any input probing rate r_I, the curve deviation $\beta_N(s/r_I, s, n)$ vanishes as probing packet size s approaches infinity. We prove this result under the condition of one-hop persistent cross-traffic routing. We also justify this conclusion informally for arbitrary cross-traffic routing and point out the major difficulty in obtaining a rigorous proof. First, we make an additional assumption as follows.

Assumption 3 *Denoting by $P_{i,\delta}(x)$ the distribution function of the δ-interval available bandwidth process $\{B_{i,\delta}(\mathbf{e}_i, t)\}$, we assume that for all $1 \leq i \leq N$, the following holds*

$$\begin{cases} P_{i,\delta}(r) = o\left(\dfrac{1}{\delta^2}\right) & r < C_i - \mathbf{x}\mathbf{e}_i \\ P_{i,\delta}(r) = 1 - o\left(\dfrac{1}{\delta^2}\right) & r > C_i - \mathbf{x}\mathbf{e}_i \end{cases}. \tag{27}$$

Recall that the mean-square ergodicity assumption we made earlier implies that as the observation interval δ gets large, the random variable $B_{i,\delta}(\mathbf{e}_i)$ converges in distribution to $C_i - \mathbf{x}\mathbf{e}_i$. Assumption 3 further ensures that this convergence is *fast* in the sense of (27). Even though this condition appears cryptic at first, it is valid in a broad range of cross-traffic environments. The next theorem shows the validity of this assumption under the condition of regenerative[9] link utilization.

Theorem 3 *When hop utilization process $\{U_i(\mathbf{e}_i, t)\}$ is regenerative, condition (27) holds.*

Note that regenerative queue is very common both in practice and in stochastic modeling literature. In fact, all the four traffic types used in [9] lead to regenerative hop workload and consequently lead to regenerative link utilization. We also conjecture that (27) holds under a much milder condition, but we leave its identification as future work.

Our next theorem states formally the convergence property of the output dispersion random variable $G_N(s/r_I, s, n)$ when s increases.

Theorem 4 *Given one-hop persistent cross-traffic routing and the three assumptions made in the paper, for any input rate r_I, the output dispersion random variable G_N of path $\mathcal{P}$ converges in mean to its fluid lower bound γ_N:*

$$\lim_{s\to\infty} E\left[G_N\left(\frac{s}{r_I}, s, n\right) - \gamma_N\left(\frac{s}{r_I}, s\right)\right] = 0. \quad (28)$$

The asymptotic variance of G_N when s increases is upper bounded by some constant K_N:

$$\lim_{s\to\infty} E\left[\left(G_N\left(\frac{s}{r_I}, s, n\right) - \gamma_N\left(\frac{s}{r_I}, s\right)\right)^2\right] \le K_N. \quad (29)$$

Note that the bounded variance, as stated in (29), is an inseparable part of the whole theorem. This is because Theorem 4 is proved using mathematical induction, where the mean convergence of G_N to γ_N can be obtained only when the mean of G_{N-1} converges to γ_{N-1} and when the variance of G_{N-1} remains bounded, as probing packet size $s \to \infty$.

We further point out that by assuming one-hop persistent cross-traffic routing, we have avoided analyzing the departure processes of cross-traffic flows. When a traversing flow of link L_i enters the path from some upstream link of L_i, the arrival process of the flow at L_i is its departure process at L_{i-1}. Unfortunately, in the queueing theory literature, there is no exact result for departure processes in FCFS queueing models if one goes beyond the assumption of Poisson arrivals. Motivated by the intractability of this problem, researchers have focused their attentions on approximations [12], [15].

To accommodate arbitrary cross-traffic routing patterns, we also need an approximation assumption which says that any cross-traffic flow that traverses link L_i (regardless of wether it enters the path from L_i or some upstream link of L_i) exhibits ergodic stationary arrival at L_i. Under this assumption, which we call "stationary departure approximation," it becomes easy to extend Theorem 4 to cover arbitrary cross-traffic routing patterns. We skip the details of this step and next apply the stationary departure approximation to examine the impact of packet-train length n on the response curve $\mathcal{Z}$.

Theorem 5 *Under the first two assumptions and the "stationary departure approximation", for any N-hop path $\mathcal{P}$ with arbitrary cross-traffic routing, for any input dispersion $g_I \in (0, \infty)$ and any probing packet size s, the random variable G_N converges to its fluid lower bound γ_N in the mean-square sense as $n \to \infty$,*

$$\lim_{n\to\infty} E\left[(G_N(g_I, s, n) - \gamma_N(g_I, s))^2\right] = 0. \quad (30)$$

Let us make several comments on the conditions of this result. First note that Assumption 3 is not necessary in this theorem. Also notice that in a single-hop path (i.e., $N = 1$), the theorem can be proved without the stationary departure approximation. However, in the multi-hop cases, the approximation is needed even when cross-traffic routing is one-hop persistent. The reason is that when n is large, the probing packet-train is also viewed as a flow, whose arrival characteristics at all but the first hop are addressed by the stationary departure approximation.

Theorem 5 shows that when the packet-train length n increases while keeping s constant, not only $E[G_N]$ converges to its fluid bound γ_N, but also the variance of G_N decays to 0. This means that we can expect almost the same output dispersion in different probings.

4.3 Discussion

Among the assumptions in this paper, some are critical in leading to our results while others are only meant to simplify discussion. We point out that the distributional stationarity assumption on cross-traffic arrivals can be greatly relaxed without harming our major results. However, this comes at the expense of much more intricate derivations. This is because when cross-traffic arrivals are allowed to be only second-order stationary or even non-stationary, the output dispersion process $\{G_N(m)\}$ will no longer be identically distributed. Consequently, the analysis of probing response curves cannot be reduced to the investigation of a *single* output dispersion random variable. Moreover, we also have to rely on an ASTA assumption on packet-train probing [9] to derive the results in this paper, which we have avoided in the present setting.

Also note that the inter-flow independence assumption is made to maintain the distributional stationarity of cross-traffic arrivals at a flow aggregation level. It only helps us avoid unnecessary mathematical rigor and is insignificant in supporting our major conclusions.

On the other hand, the mean-square ergodicity plays a central role in the (omitted) proofs for Theorem 4 and Theorem 5. A cross-traffic flow with mean-square ergodicity, when observed in a large timescale, has an almost constant arrival rate. This “asymptotically fluid like” property, is very common among the vast majority of traffic models in stochastic literature, and can be decoupled from any type of traffic stationarity. Consequently, our results have a broad applicability in practice.

Next, we provide experimental evidence for our theoretical results using testbed experiments and real Internet measurement data.

5 Experimental Verification

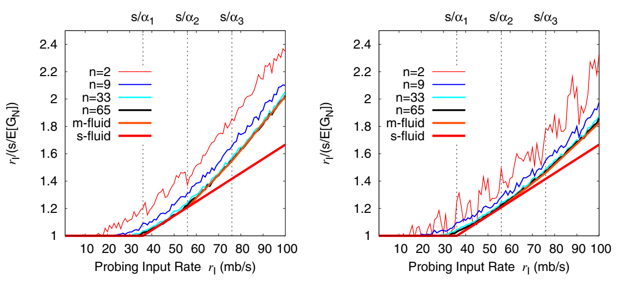
In this section, we measure the response curves in both testbed and real Internet environments. The results not only provide experimental evidence to our theory, but also give quantitative ideas of the curve deviation given in (26). To obtain the statistical mean of the probing output dispersions, we rely on direct measurements using a number of probing samples. Even though this approach can hardly produce a smooth response curve, the bright side is that it allows us to observe the output dispersion variance, reflected by the degree of smoothness of the measured response curve.

5.1 Testbed Experiments

In our first experiment, we measure in the Emulab testbed [1] the response curves of a three-hop path with the following configuration matrix (all in mb/s) and one-hop persistent cross-traffic routing

$$\mathbf{H} = \begin{pmatrix} 96 & 96 & 96 \\ 20 & 40 & 60 \end{pmatrix}. \qquad (31)$$

We generate cross-traffic using three NLANR [2] traces. All inter-packet delays in each trace are scaled by a common factor so that the average rate during the trace duration becomes the desired value. The trace durations after scaling are 1-2 minutes. We measure the average output dispersions at 100 input rates, from 1mb/s to 100mb/s with 1mb/s increasing step. For each input rate, we use 500 packet-trains with packet size 1500 bytes. The packet train length n is 65. The inter-probing delay is controlled by a random variable with sufficiently large mean. The whole experiment lasts for about 73 minutes. All three traffic traces are replayed at random starting points once the previous round is finished. By recycling the same traces in this fashion, we make the cross-traffic last until the experiment ends without creating periodicity. Also note that the packet-trains are injected with their input rates so arranged that the 500 trains



(a) one-hop persistent routing (b) path-persistent routing

Figure 2: Measured response curves using different packet train-length in the Emulab testbed.

for each input rate is evenly separated during the whole testing period.

This experiment not only allows us to measure the response curve for $n = 65$, but also for any packet-train length k such that $2 \le k < n = 65$, by simply taking the dispersions of the first k packets in each train. Fig. 2(a) shows the rate response curve $\tilde{\mathcal{Z}}(r_I, s, n)$ for $k = 2, 9, 33$ and 65 respectively. For comparison purposes, we also plot in the figure the multi-hop fluid curve $\tilde{\mathcal{F}}(r_I)$, computed from Theorem 1, and the single-hop fluid curve $\tilde{\mathcal{S}}(r_I)$ of the tight link L_3. The rate response curves $\tilde{\mathcal{Z}}(r_I, s, n)$ is defined as follows

$$\tilde{\mathcal{Z}}(r_I, s, n) = \frac{r_I}{s/E[G_N(s/r_I, s, n)]}. \qquad (32)$$

First note that the multi-hop fluid rate curve comprises four linear segments separated by turning points 36mb/s, 56mb/s, and 76mb/s. The last two linear segments have very close slopes and they are not easily distinguishable from each other in the figure. We also clearly see that the rate curve asymptotically approaches its fluid lower bound as packet-train length n increases. The curves for $n = 33$ and $n = 65$ almost coincide with the fluid bound. Also note that the smoothness of the measurement curve reflects the variance of the output dispersion random variables. As the packet train length increases, the measured curve becomes smoother, indicating the fact that the variance of the output dispersions is decaying. These observations are all in agreement with those stated in Theorem 5.

Unlike single-hop response curves, which have no deviation from the fluid bound when the input rate r_I is greater than the link capacity, multi-hop response curves usually deviate from its fluid counterpart in the entire input range. As we see from Fig. 2(a), even when the input rate is larger than 96mb/s, the measured curves still appear above $\tilde{\mathcal{F}}$. Also observe that the single-hop fluid curve $\tilde{\mathcal{S}}$ of the tight link L_3 coincides with the multi-hop fluid curve $\tilde{\mathcal{F}}$ within the input rate range $(0, 56)$ but falls below $\tilde{\mathcal{F}}$ in the input rate range $(56, \infty)$.

Finally, we explain why we choose the link capacities to be 96mb/s instead of the fast ethernet capacity 100mb/s. In fact, we did set the link capacity to be 100mb/s. However, we noticed that the measured curves can not get arbitrarily close to their fluid bound $\tilde{\mathcal{F}}$ computed based on the fast ethernet capacity. Using pathload to examine the true capacity of each Emulab link, we found that their IP layer capacities are in fact 96mb/s, not the same as their nominal value 100mb/s.

In our second experiment, we change the cross-traffic routing to path-persistent while keeping the path configuration matrix the same as given by (31). Therefore, the flow rate vector now becomes $(20, 20, 20)$.

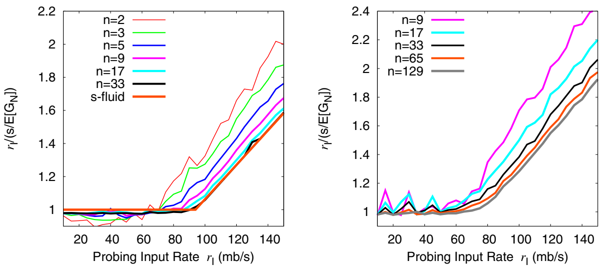
We repeat the same packet-train probing experiment and the results are plotted in Fig. 2(b). The multi-hop fluid rate curve $\tilde{\mathcal{F}}$ still coincides with $\tilde{\mathcal{S}}$ in the input rate range $(0, 56)$. When input rate is larger than 56mb/s, the curve $\tilde{\mathcal{F}}$ positively deviates from $\tilde{\mathcal{S}}$. However, the amount of deviation is smaller than that in one-hop persistent routing. The measured curve approaches the fluid lower bound $\tilde{\mathcal{F}}$ with decaying variance as packet-train length increases. For $n = 33$ and $n = 65$, the measured curves become hardly distinguishable from $\tilde{\mathcal{F}}$.

We have conducted experiments using paths with more hops, with more complicated cross-traffic routing patterns, and with various path configurations. Furthermore, we examined the impact of probing packet size using ns2 simulations, where the packet size can be set to any large values. Results obtained (not shown for brevity) all support our theory very well.

5.2 Real Internet Measurements

We conducted packet-train probing experiments on several Internet paths in the RON testbed to verify our analysis in real networks. Since neither the path configuration nor the cross-traffic routing information is available for these Internet paths, we are unable to provide the fluid bounds. Therefore, we verify our theory by observing the convergence of the measured curves to a piece-wise linear curve as packet-train length increases.

In the first experiment, we measure the rate response curve of the path from the RON node lulea in Sweden to the RON node at CMU. The path has 19 hops and a fast-ethernet minimum capacity, as we find out using traceroute and pathrate. We probe the path at 29 different input rates, from 10mb/s to 150mb/s with a 5mb/s increasing step. For each input rate, we use 200 packet-trains of 33 packets each to estimate the output probing rate $s/E[G_N]$. The whole experiment takes about 24 minutes. Again, the 200 packet-trains for each of the 29 input rates are so arranged that they are approximately evenly separated during the 24-minute testing period. The measured rate response curves associated with packet-train length 2, 3, 5, 9, 17, and 33 are plotted in Fig. 3(a), where we see that the response



(a) lulea $\longrightarrow$ CMU (b) pwh $\longrightarrow$ NYU

Figure 3: Measured response curves of two Internet paths in RON testbed .

curve approaches a piece-wise linear bound as packet-train length increases. At the same time, response curves measured using long trains are smoother than those measured using short trains, indicating the decaying variance of output dispersions. In this experiment, the curve measured using probing trains of 33-packet length exhibits sufficient smoothness and clear piece-wise linearity. We have observed two linear segments from the figure. A further investigation shows that the fluid bound of this 19-hop path only has two linear segments.

Based on (15), we apply linear regression on the second linear segment to compute the capacity C_b and the cross-traffic intensity λ_b of the tight link and get $C_b = 96$mb/s and $\lambda_b = 2$mb/s. Using these results, we retroactively plot the single-hop fluid bounds and observe that it almost overlaps with the measured curve using packet-trains of 33-packet length. Notice that the bottleneck link is under very light utilization during our 24-minute measurement period. We can also infer based on our measurement that the available bandwidth of the path is constrained mainly by the capacity of the bottleneck link and that the probing packet-trains have undergone significant interaction with cross-traffic at non-bottleneck links. Otherwise, according to Theorem 3 in [9], the response curves measured using short train lengths would not have appeared above the single-hop fluid bound when the input rate is larger than the tight link capacity 96mb/s. We believe that the tight link of the path is one of the last-mile lightly utilized fast-ethernet links and that the backbone links are transmitting significant amount of cross-traffic even though they still have available bandwidth much more than the fast-ethernet capacity. Also notice that similar to our testbed experiments, fast-ethernet links only have 96mb/s IP-layer capacity.

We repeat the same experiment on another path from the RON node pwh in Sunnyvale California to the NYU RON node. This path has 13 hops and a fast-ethernet minimum capacity. Due to substantial cross-traffic burstiness along the path, we use packet-trains of 129-packet length in our probing experiment. The other parameters such as the input rates and the number of trains used for each rate are the same as in the previous experiment. The whole measurement duration is about 20 minutes. The measured response curves are plotted in Fig. 3(b). As we see, the results exhibit more measurement variability compared to the lulea→CMU path. However, as packet-train length increases, the variability is gradually smoothed out and the response curve converges to a piece-wise linear bound. We again apply linear regression on the response curve with packet-train length 129 to obtain the tight link information. We get $C_b = 80$mb/s and $\lambda_b = 3$mb/s, which does not agree with the minimum capacity reported by pathrate. We believe that pathrate reported the correct information. Our underestimation is most probably due to the fact that there are links along the path with very similar available bandwidth. Consequently, the second linear segment become too short to detect. The linear segment we are acting upon is likely to be a latter one. This experiment confirms our analysis, at the same time shows some of the potential difficulties in exacting tight link information from the response curves.

6 Implications

We now discuss the implications of our results on existing measurement proposals. Except for pathChirp, all other techniques such as TOPP, pathload, PTR, and Spruce are related to our analysis.

6.1 TOPP

TOPP is based on multi-hop fluid rate response curve $\tilde{\mathcal{F}}$ with one-hop persistent cross-traffic routing. TOPP uses packet-pairs to measure the real rate response curve $\tilde{\mathcal{Z}}$, and assumes that the measured curve will be the same as $\tilde{\mathcal{F}}$ when a large number of packet-pairs are used. However, our analysis shows that the real curve $\tilde{\mathcal{Z}}$ is different from $\tilde{\mathcal{F}}$, especially when packet-trains of short length are used (e.g., packet-pairs). Note that there is not much path information in $\tilde{\mathcal{Z}}$ that is readily extractable unless it is sufficiently close to its fluid counterpart $\tilde{\mathcal{F}}$. Hence, to put TOPP to work in practice, one must use long packet-trains instead of packet-pairs.

6.2 Spruce

Using the notations in this paper, we can write spruce's available bandwidth estimator as follows

$$C_b\left(1 - \frac{G_N(s/C_b, s, n) - s/C_b}{s/C_b}\right), \tag{33}$$

where the probing packet size s is set to 1500bytes, the packet-train length $n = 2$, and the bottleneck link capacity C_b is assumed known.

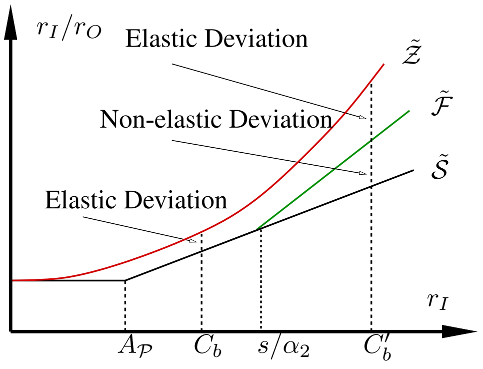


Figure 4: Illustration of two types of curve deviations.

It is shown in [9] that the spruce estimator is unbiased in single-hop paths regardless of the packet-train parameters s and n. This means that the statistical mean of (33) is equal to $A_{\mathcal{P}}$ for any $s > 0$ and any $n \geq 2$. In a multi-hop path $\mathcal{P}$, a necessary condition to maintain the unbiasedness property of the spruce estimator is

$$\tilde{\mathcal{Z}}(C_b, s, n) = \frac{\lambda_b + C_b}{C_b} = \tilde{\mathcal{S}}(C_b). \tag{34}$$

This means that at the input rate point C_b, the real rate response of path $\mathcal{P}$ must be equal to the single-hop fluid rate response at the tight link of $\mathcal{P}$.

This condition is usually not satisfied. Instead, due to Theorem 2 and Property 4, we have

$$\tilde{\mathcal{Z}}(C_b, s, n) \geq \tilde{\mathcal{F}}(C_b) \geq \tilde{\mathcal{S}}(C_b). \tag{35}$$

This implies that (33) is a negatively biased estimator of $A_{\mathcal{P}}$. The amount of bias is given by

$$C_b\Big(\tilde{\mathcal{Z}}(C_b, s, n) - \tilde{\mathcal{F}}(C_b)\Big) + C_b\Big(\tilde{\mathcal{F}}(C_b) - \tilde{\mathcal{S}}(C_b)\Big). \tag{36}$$

The first additive term in (36) is the measurement bias caused by the curve deviation of $\tilde{\mathcal{Z}}$ from $\tilde{\mathcal{F}}$ at input rate C_b, which vanishes as $n \to \infty$ due to Theorem 5. Hence we call it *elastic bias*. The second additive term is the portion of measurement bias caused by the curve deviation of $\tilde{\mathcal{F}}$ from $\tilde{\mathcal{S}}$ at input rate C_b, which remains constant with respect to the packet-train parameters s and n. Therefore it is *non-elastic*. We illustrate the two types of curve deviations in Fig. 4. Note that when $C_b < s/\alpha_2$, non-elastic bias is 0. Further recall that $s/\alpha_2 \geq A_{b2}$ as stated in Property 3. Hence, a sufficient condition for zero non-elastic bias is $C_b \leq A_{b2}$. Conceptually, elastic deviation stems from cross-traffic burstiness and non-elastic deviation is a consequence of multi-hop effects.

In Table 2, we give the amount measurement bias caused by the two types of curve deviations in both the Emulab testbed experiments and the real Internet probing measurement on the path from lulea to CMU. Note that in the

experiment	elastic bias	non-elastic bias	total bias
Emulab-1	0.56×96	0.315×96	74.4
Emulab-2	0.28×96	0.125×96	38.8
lulea-cmu	0.25×96	0	24

Table 2: Spruce bias in Emulab and Internet experiment (in mb/s).

testbed experiment using a 3-hop path with one-hop persistent routing, spruce suffers about 74mb/s measurement bias, which is twice as much as the actual path available bandwidth 36mb/s. In the second Emulab experiment using path-persistent cross-traffic, the measurement bias is reduced to 38.8mb/s, which however is still more than the actual available bandwidth. In both cases, spruce estimator converges to negative values. We used spruce to estimate the two paths and it did in fact give 0mb/s results in both cases. For the Internet path from lulea to CMU, spruce suffers 24mb/s negative bias and produces a measurement result less than 70mb/s, while the real value is around 94mb/s. We also use pathload to measure the three paths and observe that it produces pretty accurate results.

The way to reduce elastic-bias is to use long packet-trains instead of packet-pairs. In the lulea→CMU experiment, using packet-trains of 33-packet, spruce can almost completely overcome the 24mb/s bias and produce an accurate result. However, there are two problems of using long packet-trains. First, there is not a deterministic train length that guarantees negligible measurement bias on any network path. Second, when router buffer space is limited and packet-train length are too large, the later probing packets in each train may experience frequent loss, making it impossible to accurately measure $\tilde{\mathcal{F}}(C_b)$. After all, spruce uses input rate C_b, which can be too high for the bottleneck router to accommodate long packet-trains. On the other hand, note that non-elastic bias is an inherit problem for spruce. There is no way to overcome it by adjusting packet-train parameters.

6.3 PTR and pathload

PTR searches the first turning point in the response curve $\tilde{\mathcal{Z}}(r_I, s, n)$ and takes the input rate at the turning point as the path available bandwidth $A_{\mathcal{P}}$. This method can produce accurate result when the real response curve $\tilde{\mathcal{Z}}$ is close to $\tilde{\mathcal{F}}$, which requires packet-train length n to be sufficiently large. Otherwise, PTR is also negatively biased and underestimates $A_{\mathcal{P}}$. The minimum packet-train length needed is dependent on the path conditions. The current version of PTR use packet train length $n = 60$, which is probably insufficient for the Internet path from pwh to CMU experimented in this paper.

Pathload is in spirit similar to PTR. However, it searches the available bandwidth region by detecting one-way-delay increasing trend within a packet-train, which is different from examining whether the rate response $\tilde{\mathcal{Z}}(r_I, s, n)$ is greater than one [7]. However, since there is a strong statistical correlation between a high rate response $\tilde{\mathcal{Z}}(r_I, s, n)$ and the one-way-delay increasing tend within packet-trains, our analysis can explain the behavior of pathload to a certain extent. Recall that, as reported in [6], pathload underestimates available bandwidth when there are multiple tight links along the path. Our results demonstrate that the deviation of $\tilde{\mathcal{Z}}(r_I, s, n)$ from $\tilde{\mathcal{F}}$ in the input rate range $(0, A_{\mathcal{P}})$ gives rise to a potential underestimation in pathload. The underestimation is maximized and becomes clearly noticeable when non-bottleneck links have the same available bandwidth as $A_{\mathcal{P}}$, given that the other factors are kept the same.

Even through multiple tight links cause one-way-delay increasing trend for packet-trains with input rate less than $A_{\mathcal{P}}$, this is *not* an indication that the network can not sustain such an input rate. Rather, the increasing trend is a *transient* phenomenon resulting from probing intrusion residual, and it disappears when the input packet-train is sufficiently long. Hence, it is our new observation that by further increasing the packet-train length, the underestimation in pathload can be mitigated.

7 Related Work

Besides the measurement techniques we discussed earlier, Melander *et al.* [13] first discussed the rate response curve of a multi-hop network path carrying fluid cross-traffic with one-hop persistent routing pattern. Dovrolis *et al.* [3], [4] considered the impact of cross-traffic routing on the output dispersion rate of a packet-train. It was also pointed out that the output rate of a back-to-back input packet-train (input rate $r_I = C_1$, the capacity of the first hop L_1) converges to a point they call "asymptotic dispersion rate (ADR)" as packet-train length increases. The authors provided an informal justification as to why ADR can be computed using fluid cross-traffic. They demonstrated the computation of ADR for several special path conditions. Note that using the notations in this paper, ADR can be expressed as

$$\lim_{n \to \infty} \frac{s}{G_N(s/C_1, s, n)} = \frac{s}{\gamma_N(s/C_1, s)}. \qquad (37)$$

Our work not only formally explains previous findings, but also generalizes them to such an extent that allows any input rate and any path conditions.

Kang *et al.* [8] analyzed the gap response of a single-hop path with bursty cross-traffic using packet-pairs. The paper had a focus on large input probing rate. Liu *et al.* extended the single-hop analysis for packet-pairs [11] and packet-trains [9] to arbitrary input rates and discussed the impact of packet-train parameters.

8 Conclusion

This paper provides a stochastic characterization of packet-train bandwidth estimation in a multi-hop path with arbitrarily routed cross-traffic flows. Our main contributions include derivation of the multi-hop fluid response curve as well as the real response curve and investigation of the convergence properties of the real response curve with respect to packet-train parameters. The insights provided in this paper not only help understand and improve existing techniques, but may also lead to a new technique that measures tight link capacity.

There are a few unaddressed issues in our theoretical framework. In our future work, we will identify how various factors, such as path configuration and cross-traffic routing, affect the amount of deviation between $\mathcal{Z}$ and $\mathcal{F}$. We are also interested in investigating new approaches that help detect and eliminate the measurement bias caused by bursty cross-traffic in multi-hop paths.

Acknowledgements

Dmitri Loguinov was supported by NSF grants CCR-0306246, ANI-0312461, CNS-0434940.

References

[1] Emulab. http://www.emulab.net.

[2] National Laboratory for Applied Network Research. http://www.nlanr.net.

[3] C. Dovrolis, P. Ramanathan, and D. Moore, "What Do Packet Dispersion Techniques Measure?," *IEEE INFOCOM*, April 2001.

[4] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet Dispersion Techniques and a Capacity Estimation Methodology," *IEEE/ACM Transaction on Networking*, March 2004.

[5] N. Hu and P. Steenkiste, "Evaluation and Characterization of Available Bandwidth Probing Techniques," *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 3rd Quarter 2003.

[6] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *ACM SIGCOMM*, August 2002.

[7] M. Jain and C. Dovrolis, "Ten Fallacies and Pitfalls in End-to-End Available Bandwidth Estimation," *ACM IMC*, October 2004.

[8] S. Kang, X. Liu, M. Dai, and D. Loguinov, "Packet-pair Bandwidth Estimation: Stochastic Analysis of a Single Congested Node," *IEEE ICNP*, October 2004.

[9] X. Liu, K. Ravindran, B. Liu, and D. Loguinov, "Single-Hop Probing Asymptotics in Available Bandwidth Estimation: Sample-Path Analysis," *ACM IMC*, October 2004.

[10] X. Liu, K. Ravindran, and D. Loguinov, "Multi-Hop Probing Asymptotics in Available Bandwidth Estimation: Stochastic Analysis," Technical report, CUNY, Available at http://www.cs.gc.cuny.edu/tr/TR-2005010.pdf, August 2005.

[11] X. Liu, K. Ravindran, and D. Loguinov, "What Signals Do Packet-pair Dispersions Carry?," *IEEE INFOCOM*, March 2005.

[12] W. Matragi, K. Sohraby, and C. Bisdikian, "Jitter Calculus in ATM Networks: Multiple Nodes," *IEEE/ACM Transctions on Networking*, 5(1):122–133, 1997.

[13] B. Melander, M. Bjorkman, and P. Gunningberg, "A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks," *IEEE Globecom Global Internet Symposium*, November 2000.

[14] B. Melander, M. Bjorkman, and P. Gunningberg, "Regression-Based Available Bandwidth Measurements," *SPECTS*, July 2002.

[15] Y. Ohba, M. Murata, and H. Miyahara, "Analysis of Interdeparture Processes for Bursty Traffic in ATM Networks," *IEEE Journal on Selected Areas in Communications*, 9, 1991.

[16] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," *Passive and Active Measurement Workshop*, 2003.

[17] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," *ACM IMC*, 2003.

[18] W. Szczotka, "Stationary representation of queues. I.," *Advance in Applied Probability*, 18:815–848, 1986.

[19] W. Szczotka, "Stationary representation of queues. II.," *Advance in Applied Probability*, 18:849–859, 1986.

[20] R. Wolff. *Stochastic modeling and the theory of queues*. Prentice hall, 1989.

Notes

[1]In general, the tight link can be different from the link with the minimum capacity, which we refer to as the *narrow* link of $\mathcal{P}$.

[2]We use the term "fluid" and "constant-rate fluid" interchangeably.

[3]The analysis assumes infinite buffer space at each router.

[4]The term Ω_i represents the volume of fluid cross-traffic buffered between the packet-pair in the outgoing queue of link L_i. For an analogical understanding, we can view the packet-pair as a bus, the cross-traffic as passengers, and the routers as bus stations. Then, Ω_i is the amount of cross-traffic picked up by the packet-pair at link L_i as well as all the upstream links of L_i. This cross-traffic will traverse over link L_i due to the flows' routing decision.

[5]Note that the turning points in $\mathcal{F}$ is indexed according to the decreasing order of their values. The reason will be clear shortly when we discuss the rate response curve.

[6]Note that the hop available bandwidth of link L_i that is of measurement interest, given by $A_i = C_i - \mathbf{x}\mathbf{r}_i$ can be less than $C_i - \mathbf{x}\mathbf{p}$.

[7]Note that the output dispersion process can be correlated. However, this does not affect the sample-path time average of the process.

[8]See section 3.2 in [9] for more discussions about this term in a single-hop context, where R_i is referred to as *intrusion residual*.

[9]Refer to [20, pages 89] for the definition of regenerative processes.

Exploiting Internet Route Sharing for Large Scale Available Bandwidth Estimation

Ningning Hu, Peter Steenkiste
Carnegie Mellon University
$\{$*hnn, prs*$\}$*@cs.cmu.edu*

Abstract

Recent progress in active measurement techniques has made it possible to estimate end-to-end path available bandwidth. However, how to efficiently obtain available bandwidth information for the N^2 paths in a large N-node system remains an open problem. While researchers have developed coordinate-based models that allow any node to quickly and accurately estimate latency in a scalable fashion, no such models exist for available bandwidth. In this paper we introduce BRoute — a scalable available bandwidth estimation system that is based on a route sharing model. The characteristics of BRoute are that its overhead is linear with the number of end nodes in the system, and that it requires only limited cooperation among end nodes. BRoute leverages the fact that most Internet bottlenecks are on path edges, and that edges are shared by many different paths. It uses AS-level source and sink trees to characterize and infer path-edge sharing in a scalable fashion. In this paper, we describe the BRoute architecture and evaluate the performance of its components. Initial experiments show that BRoute can infer path edges with an accuracy of over 80%. In a small case study on Planetlab, 80% of the available bandwidth estimates obtained from BRoute are accurate within 50%.

1 Introduction

Recent progress in measurement techniques has made it possible to estimate path available bandwidth [11, 21, 18, 17, 23, 25]. These tools have enhanced our understanding of Internet end-to-end performance, and can be used to improve the performance of network applications. However, how to efficiently obtain available bandwidth information for the N^2 paths in a large N-node system remains an open problem. At the same time, a scalable available bandwidth estimation system has many potential applications. For example, a large service provider may want to know the available bandwidth performance for all its customers; P2P systems may want to know the available bandwidth between all node-pairs so as to select the best overlay topology; or people may want to monitor the health of a large scale system or testbed, like Planetlab [4].

Researchers have been able to build scalable systems for Internet latency estimation by using synthetic coordinated systems to significantly reduce the number of required measurements [22, 12]. Unfortunately, we do not have a similar concept for available bandwidth. Brute force solutions will not work either. The overhead to probe one path is at least around 100KB [25, 17], so measuring all end-to-end paths in a 150-node system would already require over 2GB for just one snapshot. This approach clearly does not scale to a large number of nodes, let alone to the whole Internet. Another challenge is that most available bandwidth measurement tools need to run on both ends of a network path to conduct measurements. This complicates the deployment of the tools significantly.

In this paper, we propose a scalable available bandwidth estimation system—BRoute. Here "available bandwidth" refers to the residual bandwidth left on an end-to-end path; it is determined by the available bandwidth of the bottleneck link, i.e., the link with smallest residual bandwidth. The goal of BRoute is to estimate the available bandwidth for any node-pair in a large system, with limited measurement overhead and limited cooperation among end nodes. BRoute is based on two observations. First, Hu et.al. [16] have observed that over 86% of Internet bottlenecks are within 4 hops from end nodes, i.e., on path edges. This suggests that bandwidth information for path edges can be used to infer end-to-end available bandwidth with high probability. Moreover, links near the end nodes are often shared by many paths, thus providing the opportunity to limit measurement overhead. This leads to the two key challenges in BRoute: how to measure the available bandwidth of path edges and how to quickly determine which edges a path uses.

The primary contribution of this paper is the BRoute system architecture: we discuss in Section 2 how it leverages routing information to reduce available bandwidth estimation overhead. In Sections 4–6, we first use an extensive set of measurement to show that many Internet paths exhibit the properties that BRoute relies on, we then present an algorithm that uses AS-level source and sink tree information to infer network path edges, and we finally describe how we measure the available bandwidth near end nodes. We discuss related work and conclude in Sections 7 and 8.

2 System Design

2.1 BRoute Intuition

The BRoute design is based on two important observations. First, most bottlenecks are on path edges, so for most paths we only need to obtain available bandwidth information for both edges of a path to estimate path available bandwidth.

This research was funded in part by NSF under award number CCR-0205266 and in part by KISA (Korea Information Security Agency) and CyLab Korea.

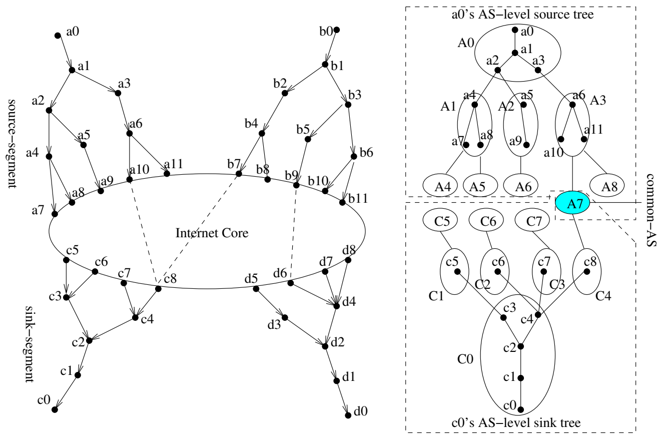


Figure 1: End-segments and AS-level source/sink trees

Second, relatively few routes exist near the source and destination compared with the core of the Internet, thus simplifying the problem of determining which edges a path takes, and which bottleneck it encounters. These observations lead to the two main operations in BRoute. First, each node collects both routing and bottleneck information for the network "edge" to which it is attached, using traceroute and Pathneck [15], respectively. This information can be published, similar to a set of coordinates. Second, in order to estimate the available bandwidth between a source node and a sink node, any node can collect the routing and bottleneck information for the source and sink and use it to determine the route taken at the edges of the path, and thus the likely bottleneck location and available bandwidth.

Before we describe BRoute in more detail, let us first define what we mean by the "edge" of a path. It corresponds to the first E and the last N links of a complete IP level path; we will call these two partial paths the *source-segment* and *sink-segment* respectively. In this paper we will use $E = 4, N = 4$ since this captures most of the bottlenecks [16]. However, different values can be used. Formally, let the path from s to d be $Path(s, d) = (r_0 = s, r_1, r_2, ..., r_n = d)$, here $r_i (1 \leq i \leq n - 1)$ are routers on the path. Then the source-segment of $Path(s, d)$ is $srcSgmt(s, d) = (r_0, r_1, r_2, r_3, r_4)$, and the sink-segment of $Path(s, d)$ is $sinkSgmt(d, s) = (r_{n-4}, r_{n-3}, r_{n-2}, r_{n-1}, r_n)$. The left graph of Figure 1 illustrates the source-segments for end nodes a_0, b_0 and the sink-segments for end nodes c_0, d_0. The dashed lines indicate the omitted central part of the paths. In this paper, we also use the term *end-segment* to indicate either a source-segment or a sink-segment.

If bottlenecks are on end-segments, we only need to consider the trees composed of the source-segments and sink-segments (called the source and sink tree), and we can ignore links within the "Internet Core" as illustrated in Figure 1. Each node can characterize both the structure and bandwidth properties of its source and sink tree. This information can be published in a central location or using a publish-subscribe system. Other nodes can then use that information to estimate the bandwidth for the paths toward or from that node. In large systems, many paths will share a same end-segment. For example, $Path(a_0, c_0)$ and $Path(b_0, c_0)$ share sink-segment $(c_8, c_4, c_2, c_1, c_0)$. This means that the measurement overhead is proportional to the number of end-segments, not the number of paths. Based on the data set discussed in Section 3, we found that, assuming source/sink trees with a depth of 4, Internet end nodes have on average only about 10 end-segments, so the overhead is linear in the number of system nodes.

Besides identifying source and sink threes, BRoute needs to identify the source-segment and sink-segment for a path without direct measurement, i.e. identifying the leaves of the trees in Figure 1. BRoute does this using AS-level path information. Intuitively, for a pair of nodes s and d, if we know all the upstream AS paths from s (called the AS-level source tree or $srcTree(s)$) and all the downstream AS paths toward d (called the AS-level sink tree or $sinkTree(d)$), then $Path(s, d)$ should pass one of their shared ASes. For example, the right graph of Figure 1 illustrates the upstream AS paths from a_0, and the downstream AS paths toward c_0. Assume that $A7$ is the only shared AS then this means that path $Path(a_0, c_0)$ must pass through $A7$, and we can use $A7$ to identify $srcSgmt(a_0, c_0)$ and $sinkSgmt(c_0, a_0)$. We will call the AS that is shared and on the actual path the common-AS. Of course, there will typically be multiple shared ASes between $srcTree(s)$ and $sinkTree(d)$. We will discuss in Section 5.1 how to uniquely determine the common-AS.

2.2 BRoute Architecture

The BRoute architecture includes three components: system nodes, traceroute landmarks, and an information exchange point. System nodes are Internet nodes for which we want to estimate available bandwidth; they are responsible for collecting their AS-level source/sink trees, and end-segment available bandwidth information. Traceroute landmarks are a set of nodes deployed in specific ASes; they are used by system nodes to build AS-level source/sink trees and to infer end-segments. An information exchange point, such as a server or publish-subscribe system, collects measurement data from system nodes and the bandwidth estimation operations can then be carried by either the server or alternatively, by the querying client. For simplicity, we will assume the use of a *BRoute server* in this paper.

BRoute leverages two existing techniques: bottleneck detection [15] and AS relationship inference [14, 26]. Bottleneck detection is used to measure end-segment bandwidth, and AS relationship information is used to infer the end-segments of a path. The operation of BRoute can be split into a pre-processing stage and a query stage (Figure 2):

- **Pre-processing:** In this stage, each system node conducts a set of traceroute measurements to the traceroute landmarks. Similarly, traceroute landmarks conduct traceroutes toward system nodes. The system node then uses the traceroute information to construct AS-level source and sink trees. Next the system node identifies its source-segments and sink-segments and

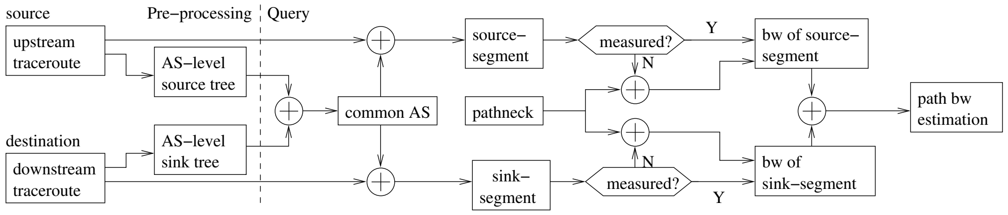


Figure 2: BRoute System Architecture

uses Pathneck to collect bandwidth information for each end-segment. This information is reported to the BRoute server.

- **Query:** Any node can query BRoute server for an estimate of the available bandwidth between two system nodes—s to d. The BRoute server will first identify the common-AS between $srcTree(s)$ and $sinkTree(d)$. The common-AS is used to identify the end-segments $srcSgmt(s,d)$ and $sinkSgmt(d,s)$ of $Path(s,d)$, and the BRoute server then returns the smaller of the available bandwidths for $srcSgmt(s,d)$ and $sinkSgmt(d,s)$ as the response to the query.

A distinguishing characteristic of BRoute is that *it uses AS-level source/sink tree computation to replace expensive network measurements*. In the following four sections, we first describe the data sets used in our analysis, and we then elaborate on three central features of BRoute: key properties of AS-level source and sink trees, end-segment inference, and end-segment bandwidth measurement.

3 Data Collection

The evaluation of the BRoute design uses five data sets:

The BGP data set includes BGP routing tables downloaded from the following sites on 01/04/2005: University of Oregon Route Views Project [8], RIPE RIS (Routing Information Service) Project [5], and the public route servers listed on [7]. These BGP tables include views from 190 vantage points, which allow us to conduct a relatively general study of AS-level source/sink tree properties.

The Rocketfuel data set is mainly used for IP-level analysis of end-segments. We use the traceroute data collected on 12/20/2002 by the Rocketfuel project [24, 6], where 30 Planetlab nodes are used to probe over 120K destinations.

The Planetlab data set was collected by the authors using 160 Planetlab nodes at different sites. It includes traceroute result from each node to all the other nodes and it is used to characterize AS-level sink tree properties.

The AS-Hierarchy data set was downloaded from [1] to match our route data sets. It contains two snapshots: one from 01/09/2003 (the closest available data set to the *Rocketfuel* data set in terms of measurement time), which is used for mapping *Rocketfuel* data set; the other from 02/10/2004, which is the latest snapshot available, and is used for mapping *BGP* and *Planetlab* data sets. This data set uses the heuristic proposed by Subramanian et.al. [26] to rank all the ASes in the BGP tables used in the computation.

The IP-to-AS data set was downloaded from [3]. Its IP-to-AS mapping is obtained using a dynamic algorithm, which is shown to be better than results obtained directly from BGP tables [20].

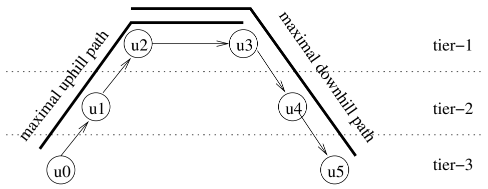


Figure 3: Maximal uphill/downhill path

4 AS-Level Source/Sink Tree

In this section, we define AS-level source/sink trees and show that they are very similar to real tree structures.

4.1 Definition

The definition of AS-level trees is based on the ranking system from [26], where all ASes are classified into five tiers. Tier-1 includes ASes belonging to global ISPs, while tier-5 includes ASes from local ISPs. Intuitively, if two connected ASes belong to different tiers, they should have a provider-to-customer or customer-to-provider relationship; otherwise, they should have a peering or sibling relationship. To be consistent with the valley-free rule [14], we say that an AS with a smaller (larger) tier number is in a *higher* (*lower*) tier than an AS with a larger (smaller) tier number. An end-to-end path needs to first go uphill from low-tier ASes to high-tier ASes, then downhill until reaching the destination (see Figure 3).

Formally, let $Tier(u_i)$ denote the tier number of AS u_i, then an AS path $(u_0, u_1, ..., u_n)$ is said to be valley-free iff there exist $i, j (0 \leq i \leq j \leq n)$ satisfying: $Tier(u_0) \geq ... \geq Tier(u_{i-1}) > Tier(u_i) = ... = Tier(u_j) < Tier(u_{j+1}) \leq ... \leq Tier(u_n)$. The maximal uphill path is then $(u_0, u_1, ..., u_j)$, and the maximal downhill path is $(u_i, u_{i+1}, ..., u_n)$. The AS(es) in the highest tier $\{u_i, ..., u_j\}$ are called *top-AS(es)*.

We can now define the *AS-level source tree* for a node s as the graph $srcTree(s) = (V, E)$, where $V = \{u_i\}$ includes all the ASes that appear in one of the maximal uphill paths starting from s, and $E = \{(u_i, u_j) | u_i \in V, u_j \in V\}$ includes the directional links among the ASes in V, i.e. $(u_i, u_j) \in E$ iff it appears in one of the maximal uphill paths starting from s. The *AS-level sink tree* is defined similarly, except that we use maximal downhill paths.

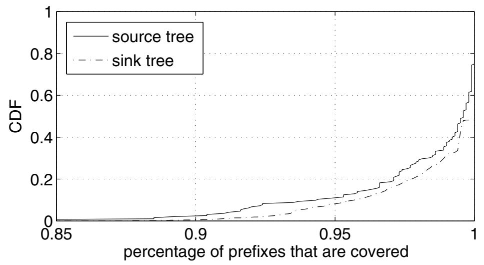


Figure 4: The tree proximities of the AS-level source/sink trees from the *BGP* data set

4.2 Tree Structure

In this subsection, we show that AS-level source/sink trees closely approximate tree structures. As we will see in the next section, this is important, since it allows BRoute to map the common-AS to a unique tree branch. There are two reasons why an AS-level source/sink tree may not be a tree. First, ASes in the same tier can have a peering or a sibling relationship, where data can flow in either direction; that can result in a loop in the AS-level source/sink tree. Second, customer-to-provider or provider-to-customer relationship can cross multiple tiers.

To study how closely AS-level source/sink trees approximate real tree structures, we define the *tree-proximity metric*. For AS-level source trees it is defined as follows; the definition for AS-level sink trees is similar. We first extract all maximal uphill paths from a data set that provides path information, for example, as obtained from BGP (*BGP* data set) or traceroute (*Rocketfuel* data set). This is done for each view point, where a view point is either a peering point (*BGP* data set) or a measurement source node (*Rocketfuel* data set). We then count the number of prefixes covered by each maximal uphill path, and use that number as the popularity measure of the corresponding maximal uphill path. Next we construct a tree by adding the maximal uphill paths sequentially, starting from the most popular one. If adding a new maximal uphill path introduces non-tree links, i.e., gives a node a second parent, we discard that maximal uphill path. As a result, the prefixes covered by that discarded maximal uphill path will not be covered by the resulting tree. The *tree proximity* of the corresponding AS-level source tree is defined as the percentage of prefixes covered by the resulting tree. While this greedy method does not guarantee that we cover the largest number of prefixes, we believe it provides a reasonable estimate on how well an AS-level source tree approximates a real tree.

Using the *BGP* data set, we can build an AS-level source tree for each of the 190 view points. The distribution of the tree proximities is shown as the solid curve in Figure 4. About 90% of the trees have a proximity over 0.95, and over 99% are above 0.88. This shows that the AS-level source trees indeed resemble real trees. This conclusion is consistent with the finding by Battista et.al. [9], who noticed that a set of AS relationships can be found to perfectly match the partial view of BGP routes from a single vantage point.

We also built AS-level sink trees using the *BGP* data set. We identified 87,877 prefixes that are covered by at least 150 view points, i.e., for which we can get over 150 maximal downhill paths. We picked the number "150" because it can give us a large number of trees. The results, illustrated by the dot-dash curve in Figure 4, are very similar with those for AS-level source trees. Actually, the tree-proximity from sink trees are slightly better, which could be a result of the limited number of downstream routes used for the AS-level sink-tree construction.

We repeated the same analysis for the *Rocketfuel* data set and reached similar conclusions. When looking at the causes for discarding a maximum uphill path during tree construction, we found that the second cause for violations of the tree property, i.e. the creation of multiple paths to reach a higher tier AS, was by far the most common reason. We speculate that these violations are caused by load-balancing-related routing policies such as MOAS (Multiple Origin AS) and SA (Selected Announced Prefixes).

As a final note, we also looked at how to efficiently measure AS-level source/sink tree. We found that if we deploy one landmark in each of the tier-1 and tier-2 ASes (totally 237 ASes in the 02/10/2004 *AS-Hierarchy* data set), we can cover at least 90% of most AS-level source/sink trees. That suggests that 200-300 landmarks can do a reasonable job.

5 End-Segment Inference

We are now ready to describe two key operations of BRoute: how to pick the common-AS, and how to use the common-AS to identify the source-segment and sink-segment of a path.

5.1 Selecting the Common-AS

Algorithm: Typically, an AS-level source tree and an AS-level sink tree share multiple ASes. We use the following algorithm to choose one of them as the common-AS. Based on the fact that most AS-level routes follow the shortest AS path [19], the algorithm first searches for the shared ASes that are closest to the root ASes in both $srcTree(s)$ and $sinkTree(d)$. If we are left with multiple candidates, we pick the one that has the highest probability to appear on $Path(s, d)$ in the measurement. We must also consider two special cases: (a) one or both root ASes can be shared, and (b) there may be no shared AS between the *measured* trees. For case (a), we return either one or both root ASes as the common-AS(es), while for case (b), we consider all ASes as shared, and we pick based on their occurrence probabilities.
Evaluation: Given the data we have, we can use two methods to evaluate the above algorithm. The first is to apply the algorithm on the AS-level source and sink trees described in Section 4.2. This method is straightforward and is used in the case study of BRoute discussed in Section 6. This method however has the drawback that it is based on limited downstream data, so the AS-level sink trees can be incomplete. In this section we use a different method: we evaluate the algorithm using the $srcTree(d)$ to replace the incomplete $sinkTree(d)$. The basis for this method is the observation that the AS-level trees are only used to determine the end-segments of a path (we do not need the AS-level path

itself), and the AS-level source tree may be a good enough approximation of the AS-level sink tree for this restricted goal. This in fact turns out to be the case in our data sets.

Using the *BGP* data set, we construct an AS-level source tree for each vantage point, infer the common-AS for each pair of vantage points, and then compare the result with the actual AS paths in the BGP tables. To make sure we have the correct path, we exclude those AS paths whose last AS is not the AS of the destination vantage point. For the 15,383 valid AS paths, the common-AS algorithm selects the wrong common-AS for only 514 paths, i.e. the success rate is 97%. Furthermore, for the 14,869 correctly inferred common-ASes, only 15 are not top AS, which confirms our intuition that the common-AS inferred is typically a top-AS, where the maximal uphill and downhill paths meet.

5.2 End-Segment Mapping

Given that the AS-level source and sink trees closely follow a tree structure, the common-AS can be easily used to identify a unique branch in both the AS-level source and sink tree. We now look at how well this AS-level tree branch can be used to determine IP-level end-segments of the path.

Ideally, for any AS $A \in srcTree(s)$, we would like to see that all upstream paths from s that pass A share the same source-segment e. If this is the case, we say A is *mapped* onto e, and every time A is identified as the common-AS, we know that the source-segment of the path is e. In practice, upstream paths from s that pass A could go through different source-segments, due to reasons such as load-balance routing or multihoming. To quantify the differences among the source-segments that an AS can map onto, we define the *coverage* of source-segments as follows. Suppose AS A is mapped to $k (k \geq 1)$ source-segments $e_1, e_2, ..., e_k$, each of which covers $n(e_i)(1 \leq i \leq k)$ paths that pass A. The coverage of e_i is then defined as $n(e_i)/\sum_{i=1}^{k} n(e_i)$. If we have $n(e_1) > n(e_2) > ... > n(e_k)$, then e_1 is called the top-1 source-segment, e_1 and e_2 are called the top-2 source-segments, etc. In BRoute, we use 0.9 as our target coverage, i.e., if the top-1 source-segment e_1 has coverage over 0.9, we say A is mapped onto e_1.

We use the *Rocketfuel* data set to analyze how many end-segments are needed to achieve 0.9 coverage. The 30 AS-level source trees built from this data set include 1687 ASes (the same AS in two different trees is counted twice), 1101 of which are mapped onto a single source-segment (i.e. coverage of 1). Among the other 586 ASes (from 17 trees) that are mapped onto multiple source-segments, 348 can be covered using the top-1 source-segments, so in total, (1101 + 348 = 1449) (85%) ASes can be mapped onto a unique source-segment. If we allow an AS to be covered using the top-2 source-segments, this number increases to 98%, i.e., only 2% (17 ASes) cannot be covered.

We used the *Planetlab* data set, which includes many downstream routes for each node, to look at the sink-segment uniqueness. We found that the above conclusion for source-segments also applies to sink-segment. Among the 99 nodes that have at least 100 complete downstream routes, 69 (70%) nodes have at least 90% of the ASes in their AS-level sink tree mapped onto top-1 sink-segments, while 95 (96%) nodes have at least 90% of their ASes mapped onto top-2 sink-segments.

Based on these end-segment properties, we map the common-AS onto top-1 or top-2 end-segments. In the first case, we return the available bandwidth of the top-1 end-segment. In the second case, we return the average of the available bandwidth of the two top-2 end-segments as the path bandwidth estimate. This method will work well if the reason for having top-2 end-segments is load balancing, since the traffic load on both end-segments is likely to be similar.

6 End-Segment Bandwidth Measurement

BRoute uses Pathneck to measure end-segment bandwidth. Although Pathneck only provides available-bandwidth upper or lower bounds for links on the path, it also pinpoints the bottleneck location, so we know if the measured path bandwidth applies to the source-segment or sink-segment. In BRoute, each node can easily use Pathneck to measure the available bandwidth bounds on its source-segments. However, to measure sink-segment bandwidth, system nodes need help from other nodes in the network.

BRoute can collect end-segment bandwidths in two modes: infrastructure mode and peer-to-peer mode. In the infrastructure mode, we use bandwidth landmarks that have high downstream bandwidth to measure the sink-segment bandwidth of system nodes. The bandwidth landmarks can use the same physical machines as the traceroute landmarks. In this mode, a system node uses its AS-level source tree to pick a subset of bandwidth landmarks. The system node will use Pathneck to measure source-segment bandwidth, using the selected bandwidth landmarks as destinations. Similarly, the bandwidth landmarks, at the request of the system, will measure the sink-segment bandwidth using the system node as Pathneck's destination. Clearly, each bandwidth landmark can only support a limited number of system nodes, but a back-of-envelop calculation shows that a bandwidth landmark with a dedicated Internet connection of 100Mbps can support at least 100K system nodes, assuming the default Pathneck configuration.

In the peer-to-peer mode, end-segment bandwidths are measured by system nodes themselves in a cooperative fashion. That is, each system node chooses a subset of other system nodes as peers to conduct Pathneck measurements, so as to cover all its end-segments. We use a simple greedy heuristic to find the sampling set. The main idea is to always choose the path whose two end nodes have the largest number of un-measured end-segments. In the *Planetlab* data set, this algorithm finds a sampling set that includes only 7% of all paths, which shows it is indeed effective. The peer-to-peer mode scales well, but it needs to support node churn, which can be complicated. Also, some system nodes may not have sufficient downstream bandwidth to accurately measure the available bandwidth on sink-segments of other system nodes.

Based on the peer-to-peer mode, we conducted a case study of BRoute on Planetlab using the design of Figure 2. The results show that over 70% of paths have a common-AS inference accuracy over 0.9, and around 80% of paths have an available bandwidth inference error within 50%. Although not perfect, these results are encouraging considering that available bandwidth is a very dynamic metric.

7 Related Work

BRoute is motivated by coordinate-based systems [22, 12] used for delay inference, but it does not explicitly construct a geometric space. BRoute instead leverages existing work on AS relationships [14, 26] as described in Section 4.1. Recent work on AS relationships identification [9, 19] could help improve BRoute. For example, [19] shows how to infer the AS path for a pair of nodes using only access to the destination node.

AS-level source/sink trees are an important part of the BRoute design. Others have also identified such tree structure, but at the IP level. For example, Broido et.al. [10] pointed out that 55% of IP nodes in their data set are in trees. Recently, Donnet et.al. [13] propose the Doubletree algorithm, which uses an IP-level tree structure to reduce traceroute redundancy in IP-level topology discovery. Using our terminology, the Doubletree is a combination of a source-node IP-level source tree and a destination-node IP-level sink tree. In contrast, BRoute leverages AS-level tree and we quantify how closely they approximate real trees.

BRoute uses Pathneck [15] for bandwidth measurements since it also provides the bottleneck location. For different application requirements, other bandwidth measurement tools [11, 21, 18, 17, 23, 25] can be used. In a broad sense, BRoute is a large-scale measurement infrastructure that uses information sharing to estimate available bandwidth efficiently. Several measurement architectures have been proposed, but they typically focus on ease of deployment and of data gathering. A good survey can be found in [2].

8 Conclusion and Future Work

In this paper, we presented the system architecture and primary operations of BRoute—a large-scale bandwidth estimation system. We explain we take advantage of Internet route sharing, captured as AS-level source/sink trees, to reduce bandwidth estimation overhead. We show that AS-level source/sink trees closely approximate a tree structure, and demonstrate how to use this AS-level information to infer end-segments, where most bottlenecks are located. In a small case study on Planetlab, 80% of the available bandwidth estimates obtained from BRoute are accurate within 50%.

Our BRoute study is only a first step towards a practical, scalable bandwidth estimation infrastructure. More work is needed in several areas. First, larger scale measurement studies are need to evaluate and refine BRoute. Second, we need to improve our understanding of several aspects of the algorithms, including end-segment inference, path available bandwidth estimation, and landmark selection. Finally and perhaps most importantly, we would like to deploy BRoute, both as a public service and as a platform for ongoing evaluation and research.

References

[1] AS hierarchy data set. `http://www.cs.berkeley.edu/~sagarwal/research/BGP-hierarchy/`.

[2] Internet measurement infrastructure. `http://www.caida.org/analysis/performance/measinfra`.

[3] IP to AS number mapping data set. `http://www.research.att.com/~jiawang/as_traceroute`.

[4] Planetlab. `https://www.planet-lab.org`.

[5] RIPE RIS (Routing Information Service) Raw Data. `http://www.ripe.net/projects/ris/rawdata.html`.

[6] Rocketfuel Data Sets. `http://www.cs.washington.edu/research/networking/rocketfuel`.

[7] Route Server Wiki. `http://www.bgp4.net/cgi-bin/bgp4wiki.cgi?Route_Server_Wiki`.

[8] University of Oregon Route Views Project. `http://www.routeviews.org`.

[9] G. D. Battista, M. Patrignani, and M. Pizzonia. Computing the types of the relationships between autonomous systems. In *Proc. IEEE INFOCOM*, April 2003.

[10] A. Broido and k. clafy. Internet topology: Connectivity of ip graphs. In *Proc. SPIE International Symposium on Convergence of IT and Communication*, 2000.

[11] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical report, Boston University Computer Science Department, March 1996.

[12] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proc. ACM SIGCOMM*, August 2004.

[13] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS*, June 2005.

[14] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Trans. Networking*, December 2001.

[15] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *Proc. ACM SIGCOMM*, August 2004.

[16] N. Hu, O. Spatscheck, J. Wang, and P. Steenkiste. Optimizing network performance in replicated hosting. In *The Tenth International Workshop on Web Caching and Content Distribution (WCW 2005)*, September 2005.

[17] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.

[18] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *Proc. ACM SIGCOMM*, August 2002.

[19] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang. On AS-level path inference. In *to appear in SIGMETRICS'05*, June 2005.

[20] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-level Traceroute Tool. In *Proc. ACM SIGCOMM*, September 2003.

[21] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proc. IEEE GLOBECOM*, November 2000.

[22] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE INFOCOM*, June 2002.

[23] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. PAM*, April 2003.

[24] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.

[25] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. ACM IMC*, October 2003.

[26] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *Proc. IEEE INFOCOM*, June 2002.

Inferring and Debugging Path MTU Discovery Failures

Matthew Luckie
University of Waikato
mjl@wand.net.nz

Kenjiro Cho
Internet Initiative Japan
kjc@iijlab.net

Bill Owens
NYSERNet
owens@nysernet.org

Abstract

If a host can send packets larger than an Internet path can forward, it relies on the timely delivery of Internet Control Message Protocol (ICMP) messages advising that the packet is too big to forward. An ICMP Packet Too Big message reports the largest packet size – or Maximum Transmission Unit (MTU) – that can be forwarded to the next hop. The iterative process of determining the largest packet size supported by a path by learning the next-hop MTU of each MTU-constraining link on the path is known as Path MTU Discovery (PMTUD). It is fundamental to the optimal operation of the Internet. There is a perception that PMTUD is not working well in the modern Internet due to ICMP messages being firewalled or otherwise disabled due to security concerns. This paper provides a review of modern PMTUD failure modes. We present a tool designed to help network operators and users infer the location of a failure. The tool provides fairly detailed information about each failure, so the failure can be resolved. Finally, we provide data on the failures that occurred on a large jumbo-capable network and find that although disabling ICMP messages is a problem, many other failure modes were found.

1 Introduction

Given a volume of data to send, it is desirable to encapsulate the data in the fewest number of packets possible, as "much of the cost of packetised communication is per-packet rather than per-probe" [1]. To send the fewest number of packets possible, a host must determine the largest IP packet size – or Maximum Transmission Unit (MTU) – supported by the path. The iterative process to determine the largest possible MTU on an end-to-end path by consecutively learning the next-hop MTU of each MTU-constraining link on the path is known as Path MTU Discovery (PMTUD). PMTUD allows a host or application to determine the largest IP packet size supported by an Internet path, and thus send the fewest number of packets.

Path MTU Discovery is documented in RFC 1191 for IPv4 [2] and RFC 1981 for IPv6 [3]. An application or kernel determines the largest supported MTU on an Internet path in an iterative manner, starting with the outgoing interface's MTU. It reduces the Path MTU each time a Packet Too Big (PTB) message is received until the destination host is reached, using the next-hop MTU value included in each successive PTB message. When this approach to PMTUD works, it allows an end host to quickly determine the Path MTU. There are, however, a number of well-known limitations of this technique [4], and work is in progress in the IETF to redefine the PMTUD method. This work discusses the current approach to PMTUD.

The failure modes of PMTUD are often difficult to debug, as they are triggered by relatively large packets. For example, a TCP connection may be established through a path where a PMTUD failure exists, as the TCP three-way handshake involves small packets that are unlikely to trigger a PMTUD failure. However, a PMTUD failure is likely to occur when either end of the TCP connection attempts to send a packet that is larger than can be forwarded through the path without fragmentation. A scenario like this is likely to cause the TCP connection to stall for some period of time before either failing, sending smaller packets, or allowing retransmitted packets to be fragmented.

This work introduces a technique for inferring and debugging PMTUD failures which occur on the forward path. Our technique uses a traceroute-like method to infer the location of a failure and the maximum packet size which can be forwarded through it. The technique does not infer failures that occur on the reverse path, such as the over-zealous firewalling of all inbound ICMP packets – including PTB messages – in order to protect a machine from security concerns related to ICMP or crude Denial of Service (DoS) attacks [5]. A recent study on the TCP behaviour of web-servers [6] found that PMTUD on the reverse path failed for 17% of 81776 targets tested and 35% of 500 popular web-sites tested – presumably because of middle-boxes which blocked inbound ICMP to the web-servers.

The rest of this paper is organised as follows. We begin by reviewing some of the known PMTUD failures in Section 2. We then discuss the debugging techniques used in this work to infer the location and mode of a PMTUD failure, and discuss the implementation of these techniques in our publicly available tool, scamper, in Section 3. In Section 4, we discuss the data collection that we did in support of this work, and then present some analysis of the results obtained in Section 5. Finally, we discuss a few anecdotes of strange behaviours we observed separate to the data collection for this study, before presenting our conclusions.

2 Path MTU Discovery Failure Modes

2.1 Router Configuration Issues

The most well known PMTUD failure mode is the ICMP Black Hole discussed in RFC 2923 [4]. The ICMP Black Hole problem has two halves; routers which do not send PTB messages due to misconfiguration or implementation bugs, and hosts which do not receive PTB messages due to a middle-box or firewall filtering them. The problem of router misconfiguration was first documented in RFC 1435 [7], where it was reported that code had been added to some routers to provide the capability to disable ICMP message generation in order to protect old BSD hosts, which were faulty in their handling of some ICMP messages. The RFC recommended that router code be updated to exclude PTB messages from suppression, as that particular message type did not trigger the faulty behaviour. However, it appears that this recommendation has either not been widely implemented, or operators are not using it. In the modern Internet, a router which does not send any ICMP message is almost certainly configured that way due to security concerns.

2.2 MTU Mismatches

An MTU mismatch occurs when a router and the path to the next-hop do not have a consistent understanding of the MTU. Specifically, a router believes that the path to the next hop is capable of forwarding packets larger than it actually can. Such a mismatch causes PMTUD to fail because the MTU change occurs below the IP layer, where a PTB message is not sent. A common scenario where this occurs is connecting a jumbo-capable gigabit Ethernet interface and a non-jumbo interface, which could be gigabit or fast Ethernet, across a switch. It can also occur if two jumbo interfaces are connected to a switch that does not support jumbo packets. The jumbo-capable Ethernet interface can send packets larger than 1500 bytes to the switch. However, the switch either cannot accept these packets, or cannot forward them to the next interface, and so the packets are silently discarded.

2.3 No Suggested Next-Hop MTU

The original IPv4 ICMP protocol [8] did not define the next-hop MTU field that PMTUD relies on to determine the largest packet size supported to the next hop. The next-hop MTU field was first defined in RFC 1191 [2], and makes use of otherwise unused space in the ICMP message. Routers that do not set the next-hop MTU field in a PTB message are easily detected, as the unused space is set to zero. In the face of a PTB message without a suggested next-hop MTU, current practice in the NetBSD kernel – among others – is to determine the size of the packet that caused the PTB message by examining the length field returned with the IP header embedded in the PTB message and then select a smaller packet size from a table of known MTU values.

2.4 Private Addressing

Some operators choose to use RFC 1918 [9] private addresses when numbering router interfaces in order to avoid using public addresses. The use of RFC 1918 addresses can cause PMTUD to fail if PTB messages are sent with an RFC 1918 source address, since packets with RFC 1918 source addresses are often dropped by ingress filters at the network edge.

2.5 Unspecified Implementation Bugs

There are other possibilities of PMTUD failure modes related to implementation bugs. For example, a router may send a PTB message with a suggested next-hop MTU larger than the size of the packet which caused it to be sent. Possible causes of this failure mode include not sending the next-hop MTU field in network byte order, or a router not adjusting internal state correctly when adding or removing headers. Other possible implementation bugs include: sending a PTB message with the embedded IP packet modified in some way such that the PTB message is unable to be matched with an active connection or application; sending an ICMP error message without generating a valid ICMP checksum; and sending an ICMP error message that is not a PTB message when it should have been.

3 Debugging Techniques

We have implemented two forward path debugging techniques into scamper, our publicly available measurement tool. The initial goal of the PMTUD code in scamper was to enable the detection of IPv6-over-IPv4 tunnels when comparing IPv4 and IPv6 paths between pairs of dual-stack nodes [10]. The code has evolved beyond this requirement, in part due to experiences in inferring tunnels in uncooperative paths.

To begin with, scamper conducts a standard traceroute with small UDP probes to unused ports. The purpose of this initial phase is to infer the forward IP path topology, determine which routers will provide ICMP feedback to small TTL-limited probes, and ensure that small probes are terminated somewhere in the path by an ICMP Destination Unreachable message so that scamper can distinguish between large probes being silently discarded and all probes being silently discarded. After the traceroute completes, scamper begins a PMTUD phase, where it solicits PTB messages in response to large probes until the destination is reached. scamper infers that PMTUD has failed when it does not obtain an expected reply packet to a probe the size of the currently known Path MTU value. When a PMTUD failure is detected, it uses one of two debugging techniques to infer the location of the failure and the largest packet which can be forwarded. Before we describe the two debugging techniques in detail, we describe the process by which the next-hop MTU is inferred.

3.1 Next-hop MTU Search

The purpose of the next-hop MTU search is to infer the largest packet size which can be forwarded to the next-hop. The general strategy is to, as quickly as possible, reduce a search space bounded by the smallest packet size to obtain a valid response and the largest packet size to not obtain a valid response, to find the underlying next-hop MTU. A binary search is not well suited to this task, for two reasons. First, MTU values tend to cluster due to the fairly limited combinations of media MTU values and encapsulations commonly used. Second, each probe that is discarded without the source receiving any ICMP feedback incurs a timeout delay that is often at least an order of magnitude larger than the delay incurred when probing with a packet that does obtain ICMP feedback. By default, scamper will retry a probe that obtains no ICMP feedback once, five seconds after sending the initial probe. In this scenario, a choice of probe size that does not obtain ICMP feedback incurs a ten second penalty before a different probe size can be tried. In order to determine the actual next-hop MTU as quickly and efficiently as possible, scamper is pre-loaded with a table of known MTU values.

When scamper begins a next-hop MTU search, it defines the lower bound by selecting an MTU in the table smaller than the failed probe, depending on three criteria. First, if the failed probe is larger than 1500 bytes, then scamper tries with a 1500 byte packet, as Ethernet is ubiquitous and likely to be the cause of an MTU restriction from larger frame sizes. Second, if the failed probe is larger than 1454 bytes, then scamper tries with a 1454 byte probe because 1454 is a lower bound of a series of MTU values that indicate some tunnel or encapsulation of IP over Ethernet. Otherwise, scamper selects the largest MTU from the table that is smaller than the size of the failed probe. The search for the initial lower bound is complete when ICMP feedback is obtained; the upper bound is reduced each time a probe for the initial lower bound does not obtain feedback.

After the lower bound is set, scamper then narrows the search space until it converges on the actual next-hop MTU. The approach to choosing a suitable probe size consists of three criteria, which are checked in order until a matching condition is found. First, if the lower bound of the search space is 1500 bytes or is a known MTU value in the table, and the upper bound is smaller than the next largest known MTU, then scamper probes with a packet one byte larger than the lower bound. The rationale for this is that if the search space is narrowed to within two entries in the MTU table, then there is a fair chance that the actual next-hop MTU is the current lower bound, and we can confirm this by sending a probe one byte larger. Second, if the next largest MTU in the table is smaller than the current upper bound, then scamper chooses this MTU as its next probe size. The rationale for this decision is that scamper can quickly determine the next-hop MTU if it is one of the values in the table. Lastly, if scamper is working within two known MTU values, then it will resort to a binary search to determine the next-hop MTU.

3.2 Inferring MTU without Feedback

This technique is used to infer the next-hop MTU and location of a hop that does not send PTB messages when it should. This technique is used when scamper does not obtain ICMP feedback with large packets the size of the current working Path MTU value. The technique consists of two stages. The first stage is a next-hop MTU search to infer the largest packet that can be forwarded, as described in Section 3.1. The second stage is a Time-to-Live (TTL) or Hop-Limit (HLIM) search of the forward path to infer the hop where large packets are silently discarded by determining the largest TTL or HLIM value that can be set in the IP header which still obtains an ICMP Time Exceeded message in response. This debugging technique is illustrated in Figure 1. This technique can infer a series of failure modes which are difficult to distinguish from each other, as there are many reasons why a source host may not receive a PTB message, and we have incomplete information to definitively infer why. We can, however, use a few heuristics to narrow the failure modes down.

If the farthest hop from which we obtain an ICMP Time Exceeded message with a large TTL-limited probe is immediately before a hop from which we obtain no ICMP Time Exceeded messages, we infer that the failure is likely to occur at the next hop either because all ICMP messages are disabled, or all ICMP responses from the router are being filtered somewhere in the network, possibly due to the use of RFC 1918 addresses. If we are able to receive ICMP

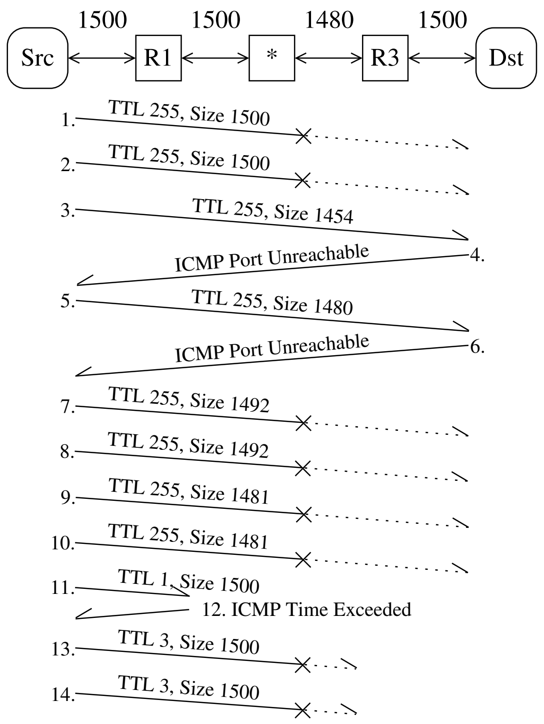


Figure 1: Inferring the MTU without feedback. An ICMP Black Hole exists between routers R1 and R3 where the MTU is restricted to 1480 bytes. A PMTUD failure is detected with probes 1 and 2, probes 3 to 10 infer that the next-hop MTU is 1480, and probes 11 to 14 infer that the large packets are probably being discarded at hop 2.

Time Exceeded messages with small TTL-limited probes from adjacent hops but we only receive Time Exceeded messages with large probes from the first hop in the path, we infer that the failure mode is likely to be either due to an interface being configured to not send any ICMP Destination Unreachable messages, or an MTU mismatch between the adjacent routers, or the PTB message originating from a different interface than the interface that sends Time Exceeded messages – with a source address that causes the PTB message to be subsequently filtered.

3.3 Inferring MTU with Invalid Feedback

This technique is used when a PTB message is received in response to a large probe, but the next-hop MTU included in the PTB message is either not set, or is larger than the probe which triggered the message. This technique uses a variation of the next-hop MTU search technique described in Section 3.1; instead of using the absence of a PTB message to reduce the upper-bound of the search space, this technique uses the faulty PTB message. This method can converge on the actual next-hop MTU fairly rapidly if ICMP feedback is received for packets smaller than the next-hop MTU past the faulty router, as the test for each probe size costs one round-trip-time (RTT). We use a slightly different technique if the path does not provide ICMP feedback after the faulty router due to another failure further in the path. When this occurs, scamper works progressively downwards through the MTU table soliciting faulty PTB messages rather than moving progressively upwards, as it would normally do. This is because scamper has to time-out on a probe which does not obtain ICMP feedback before it can send another probe, which has a much larger cost than sending packets which trigger faulty PTB messages.

3.4 Limitations

As the techniques we described rely on ICMP messages as feedback, they can be unreliable when ICMP rate-limiting is encountered. By default, scamper will send each probe twice before trying another probe type, with a five second timeout between each attempt. If two successive probes do not receive ICMP feedback due to rate-limiting, we may infer an incorrect next-hop MTU, or infer the wrong location of a failure, or infer a failure where one does not exist.

4 Methodology

We collected PMTUD failure data from two IPv4 hosts with 9000-byte MTU interfaces connected to networks that peer with Internet2, which itself is 9000-byte clean through the core, on April 28th 2005. The first location was from NYSERNet in New York, and the second was an Internet2 measurement machine in Chicago. The target list consists of 147 NLANR AMP machines, which are typically either on university campuses connected to the Internet2 network, or connected to networks that peer with Internet2. Most of the AMP machines connect to their host network with an Intel Pro100 Ethernet interface, which is capable of sending 1500 byte IP packets. Some have Gigabit Ethernet interfaces which are capable of sending IP packets larger than 1500 bytes, but are not configured to do so. The purpose of this dataset is to understand PMTUD failures on networks that can natively carry jumbo packets, and thus will require fragmentation at least at the edge of the campus network closer to each individual AMP machine.

5 Results

Of the 147 AMP machines in each dataset, we were able to complete a traceroute to at least 134 machines, or 91% of the target list. However, we inferred a PMTUD failure for 30% of the reachable machines. A summary of the failures is presented in Table 1. We categorised the failures into four groups: failure points where no ICMP messages

Dataset:	NYSERNet-east	nms1-chin	Intersection	Total
Location:	New York, NY	Chicago, IL	–	–
Hostname:	east.nysernet.org	nms1-chin.abilene.ucaid.edu	–	–
Date / Time:	Apr 28 2005, 21:50 EDT	Apr 28 2005, 20:10 CDT	–	–
Target Count:	147	147	147	–
Reachable:	136 (92.5%)	134 (91.2%)	134	–
PMTUD Failures:	41 (30.1%)	40 (29.9%)	25	–
No ICMP messages:	6 (6 unique)	5 (5 unique)	4 (4 unique)	7 unique
No PTB messages:	26 (17 unique)	27 (18 unique)	13 (13 unique)	22 unique
Incorrect PTB messages:	2 (2 unique)	2 (2 unique)	2 (2 unique)	2 unique
Target MTU Mismatch:	7 (7 unique)	6 (6 unique)	6 (6 unique)	7 unique

Table 1: Summary of the two data collections. 30% of reachable targets had a PMTUD failure.

are received (7), failure points where no PTB message is received (22), failure points where a PTB message is received with an incorrect next-hop MTU (2), and target machines which have an MTU mismatch with a router on their subnet (7). We identify a failure point by the IP addresses either side of the fault in the IP path. For example, the failure point would be identified as being between R1 and R3 in Figure 1. For each fault, we approached the technical and administrative contacts for the relevant AMP machine if the fault was determined to be local to that campus, or the operators of the relevant transit network.

We inferred seven failure points from which we did not receive any ICMP messages; of these, six were at routers where the next-hop MTU was inferred to be 1500 bytes, while the seventh had a next-hop MTU of 1536 bytes. One failure appeared to be caused by two successive routers in the path that both sent ICMP messages with a source address of 127.0.0.1, which were then discarded by a filter close to both of our measurement hosts. Similarly, another router located at the campus border used RFC 1918 addresses to number its interfaces, which also caused all ICMP messages from it to be filtered out. Another failure was caused by a BGP routing issue that, despite the fact that end-to-end connectivity was available, a significant portion of the routers on the forward path had no route back to the source host. This included one router which was therefore unable to send a PTB message to the source to signal that it was sending packets which were too big to forward. Finally, one other was due to a firewall designed to protect systems from security exploits by blocking all packets with a source or destination address matching particular addresses, including the addresses of core routers.

We found 22 hops from which we received ICMP Time Exceeded messages, but did not receive PTB messages when it was inferred that we should have. Sixteen of these hops had a next-hop MTU of 1500 bytes, accounting for just over two-thirds of the failures. Due to the method of counting hops where a failure occurs, the actual number of unique failure locations is a little less, as there is some repetition in the source address of some failure points. We determined that there were 20 failure locations. Two points were upgraded before a diagnosis could be obtained. We obtained a technical diagnosis of each fault for seven failures; three reported that they had disabled ICMP Destination Unreachable messages, while the other four were the result of an MTU mismatch or misconfiguration. For the 11 other failures for which we do not have a technical diagnosis, we probed the particular routers with a UDP probes to unused ports, in order to determine if they had disabled Destination Unreachable messages or not. Eight systems did not reply with a Destination Unreachable message.

We found two hops at one location from which we received a PTB message, but the next-hop MTU reported in the message was incorrect. The particular router would send a PTB message with a suggested next-hop MTU of 4586. It was, however, unable to forward packets larger than 4472 bytes to the next hop.

Seven targets were inferred to be on a subnet where nodes did not have a consistent agreement regarding the MTU. Two of the seven AMP targets with an MTU mismatch were able to receive IP packets larger than 1500 bytes, despite their use of 1500 byte MTU interfaces. One was able to receive packets up to 2016 bytes, while the other was able to receive packets up to 1506 bytes. We established that IP packets were arriving complete at these monitors by examining the probe packets with tcpdump.

6 Two Anecdotes

As discussed in Section 3.3, we implemented a technique to infer the correct next-hop MTU when a router sends a PTB message with an invalid next-hop MTU. The data included in this paper did not include such a failure, although we encountered one when implementing our tool. The router in question was located in New York City in the network of a large Internet Service Provider. For packet sizes between 4458 and 4470 bytes, the router would return a PTB message with an invalid next-hop MTU of 4470. Initial attempts to determine the cause of what appeared to be a bug were difficult. Initially, we were told the fault was somehow related to the next-hop having an MPLS header with room for three 4-byte MPLS labels. It was also suggested that the fault could be a particular known router bug, although the bug number suggested seems unrelated. At this time we have been unable to determine the cause of the fault, and are pursuing this matter with a router vendor.

Unspecified router bugs can also prevent PMTUD from succeeding, as discussed in Section 2.5. During the course of scamper's development, we found an IPv6 router which appeared to route IPv6 packets over an IPv6-in-IPv4 tunnel with an MTU of 1480 bytes. However, for IPv6 packets larger than 1480 bytes, we did not receive any PTB messages. Rather, it sent two Destination Unreachable, No Route messages. The first message was returned with the IPv6 probe packet intact and caused scamper to cease PMTUD to the target beyond it. The second message – which we picked up by accident while monitoring all ICMPv6 packets into the machine – was unable to be matched to any probe we sent, as the encapsulated probe packet had the source and destination port fields zeroed out. We contacted the site responsible and reported the fault. To our knowledge, the fault was never identified and corrected, and went away when the particular path was replaced with a native IPv6 path.

7 Conclusion

The consensus is that Path MTU Discovery – in its current form – is unreliable due to it relying on the timely delivery of PTB messages, which are disabled or firewalled in many networks. We hypothesise that these failures go unnoticed in routine operational testing and monitoring, as they are only noticeable with larger probe packets. The default size of probe packets sent using traceroute and ping is too small to trigger PMTUD failures, and in the absence of packet loss with these basic connectivity measures, it is tempting to declare a path as fully operational.

In this paper, we presented a series of debugging techniques which infer PMTUD failures on the forward path. Using our implementation, we collected data on PMTUD failures found in jumbogram-capable networks. We found that of the reachable targets, 30% had a failure that would prevent efficient end-to-end communication from taking place. Less than half of these failures were caused by a configuration decision to disable the ICMP messages that are necessary for PMTUD to work. As the Internet MTU is raised, particularly as jumbo-capable Ethernet interfaces become more commonplace and jumbo transit services are offered, it seems likely that the classical PMTUD methods will continue to be strained. Until the new approach to PMTUD is completed and widely deployed amongst end-hosts, we believe our tool is a useful operational utility.

Acknowledgements

scamper's development was generously funded by the WIDE project in association with CAIDA from April 2004 to March 2005. The NLANR Measurement and Network Analysis Group (NLANR/MNA) is supported by the National Science Foundation (NSF) under cooperative agreement no. ANI-0129677. Matt Zekauskas (Internet2) collected the nms1-chin dataset. Maureen C. Curran and Joe Groff provided valuable editorial assistance. Matt Brown, Nevil Brownlee, Alan Holt, and Perry Lorier provided useful feedback on the paper.

References

[1] C.A. Kent and J.C. Mogul. Fragmentation considered harmful. *ACM SIGCOMM Computer Communication Review*, 17(5):390–401, 1987.

[2] J. Mogul and S. Deering. Path MTU Discovery. RFC 1191, IETF, November 1990.

[3] J. McCann, S. Deering, and J. Mogul. Path MTU Discovery for IP version 6. RFC 1981, IETF, August 1996.

[4] K. Lahey. TCP problems with Path MTU Discovery. RFC 2923, IETF, September 2000.

[5] R. van den Berg and P. Dibowitz. Over-zealous security administrators are breaking the Internet. In *Proceedings of LISA '02: Sixteenth Systems Administration Conference*, pages 213–218, Berkeley, CA, November 2002.

[6] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. *ACM SIGCOMM Computer Communication Review*, 35(2):37–52, April 2005.

[7] S. Knowles. IESG advice from experience with Path MTU Discovery. RFC 1435, IETF, March 1993.

[8] J. Postel. Internet Control Message Protocol. RFC 792, IETF, September 1981.

[9] Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, and E. Lear. Address allocation for private internets. RFC 1918, IETF, February 1996.

[10] K. Cho, M. Luckie, and B. Huffaker. Identifying IPv6 network problems in the dual-stack world. In *Proceedings of the ACM SIGCOMM workshop on Network troubleshooting: research, theory and operations practice meet malfunctioning reality*, pages 283–288, Portland, OR., September 2004.

Characterization and Measurement of TCP Traversal through NATs and Firewalls [*]

Saikat Guha Paul Francis
Cornell University
Ithaca, NY 14853
{*saikat, francis*}*@cs.cornell.edu*

Abstract

In recent years, the standards community has developed techniques for traversing NAT/firewall boxes with UDP (that is, establishing UDP flows between hosts behind NATs). Because of the asymmetric nature of TCP connection establishment, however, NAT traversal of TCP is more difficult. Researchers have recently proposed a variety of promising approaches for TCP NAT traversal. The success of these approaches, however, depend on how NAT boxes respond to various sequences of TCP (and ICMP) packets. This paper presents the first broad study of NAT behavior for a comprehensive set of TCP NAT traversal techniques over a wide range of commercial NAT products. We developed a publicly available software test suite that measures the NAT's responses both to a variety of isolated probes and to complete TCP connection establishments. We test sixteen NAT products in the lab, and 93 home NATs in the wild. Using these results, as well as market data for NAT products, we estimate the likelihood of successful NAT traversal for home networks. The insights gained from this paper can be used to guide both design of TCP NAT traversal protocols and the standardization of NAT/firewall behavior, including the IPv4-IPv6 translating NATs critical for IPv6 transition.

1 Introduction

Network address and port translators (NATs) and firewalls break the IP connectivity model by preventing hosts outside the *protected network*[1] from initiating a connection with a host inside the protected network. If both endpoints are protected by their respective NAT or firewall, ordinary TCP cannot be established since the end initiating the TCP is outside the other end's NAT[2]. This is true even if the connection would be allowed according to each end's firewall security policy. For instance, if the firewall policy is that internal hosts may initiate TCP connections, and both hosts wish to initiate. Recent work has proposed work-arounds that establish a TCP connection without the use of proxies or tunnels [9, 5, 3, 6]. This is accomplished by setting up the necessary connection-state on the NAT through a carefully crafted exchange of TCP packets. However, not all NATs in the wild react the same way, causing these approaches to fail in various cases. Understanding such behavior in NATs and measuring how much they detract from the original goal of universal connectivity in the Internet is crucial to integrating them cleanly into the architecture.

The Internet architecture today is vastly different from that envisioned when TCP/IP was designed. Firewalls and NATs often make it impossible to establish a connection even if it does not violate policy. For instance, Alice and Bob may disallow unsolicited connections by hiding behind a NAT or configuring their firewalls to drop inbound SYN packets. Yet when both Alice and Bob agree to establish a connection there is no way to do so without reconfiguring their NAT since Alice's SYN is dropped by Bob's NAT and vice versa. Even so, NATs and firewalls have become a permanent part of the network infrastructure and will continue to remain so for a long time. Even if IPv6 is deployed globally, IPv4-IPv6 NATs will be needed during the lengthy transition, and IPv6 firewalls will be needed for security. As a result, mechanisms that enable two consenting hosts behind NATs to communicate with each other are needed.

This problem has been solved for UDP by STUN [15]. In STUN, Alice sends a UDP packet to Bob. Although this packet is dropped by Bob's NAT, it causes Alice's NAT to create local state that allows Bob's response to be directed to Alice. Bob then sends a UDP packet to Alice. Alice's NAT considers it part of the first packet's flow and routes it through, while Bob's NAT considers it a connection initiation and creates local state to route Alice's responses. This approach is used by Skype, a popular VoIP application [2]. Unfortunately, establishing TCP is more complicated. Once Alice sends her SYN packet, her OS stack

[*]This work is supported in part by National Science Foundation grant ANI-0338750

as well as her NAT expect to receive a SYNACK packet from Bob in response. However, since the SYN packet was dropped, Bob's stack doesn't generate the SYNACK. Proposed workarounds to the problem are complicated, their interactions with NATs in the wild are poorly understood, and the extent to which they solve the problem is not known. Consequently applications such as the file-transfer module in Skype use contraindicated protocols like UDP. While such approaches may work, we believe it is important that wherever possible, applications use the native OS TCP stack. This is in part to avoid increasingly complex protocol stacks, but more importantly because TCP stacks have, over the years, been carefully optimized for high performance and congestion friendliness.

Overall this work makes four contributions. First, we identify and describe the complete set of NAT characteristics important to TCP NAT traversal. Second, we measure the prevalence both of these individual characteristics and of the success rate of peer-to-peer TCP connections for the various proposed approaches. Third, based on these measurements, we suggest modifications to the proposed approaches. Fourth, we provide public-domain software toolkit that can be used to measure NATs as they evolve, and as the basis of TCP NAT traversal in P2P applications. Altogether we provide insights for application developers into the inherent tradeoffs between implementation complexity and NAT-traversal success rate. Finally, our results can be used to guide the standardization process of NATs and firewalls, making them more traversal friendly without circumventing security policies.

The rest of the paper is organized as follows. Section 2 discusses the proposed TCP NAT-traversal approaches. Section 3 and Section 4 explain our setup for testing NATs and the observed NAT behavior. Section 5 analyzes port-prediction and Section 6 analyzes peer-to-peer TCP establishment. Section 7 discusses related work. Section 8 concludes the paper.

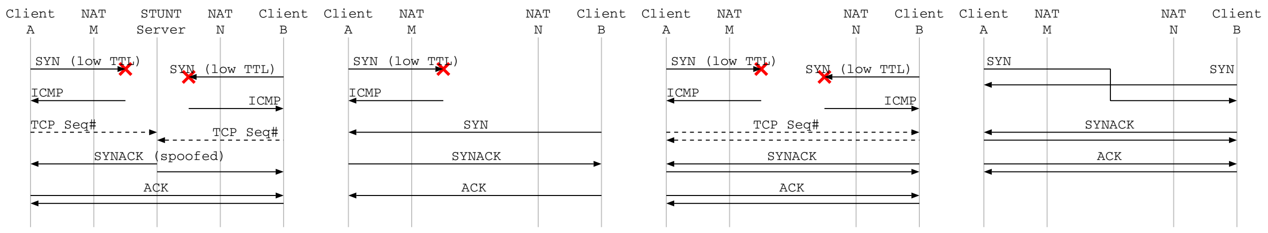
2 TCP NAT-Traversal

In this section we discuss the TCP NAT-traversal approaches that have been proposed in recent literature. In all the approaches, both ends initiate a TCP connection. The outbound SYN packet from each host creates the necessary NAT state for its own NATs. Each approach then reconciles the two TCP attempts into a single connection through different mechanisms as described in this section. The address and port to which these original SYNs are sent is determined through *port prediction*. Port prediction allows a host to guess the NAT mapping for a connection before sending the outbound SYN and is discussed in detail later. The approaches also require some coordination between the two hosts. This is accomplished over an out-of-band channel such as a connection proxied by a third party or a UDP/STUN session. Once the direct TCP connection is established, the out-of-band channel can be closed. The reconciliation mechanism used triggers different behavior in different NATs causing the proposed approaches to fail in many instances. In addition, it is possible for either endpoint to be behind *multiple NATs* in serial[3]. In such cases the behavior observed is a composite of the behavior of all the NATs and firewalls in the path. For brevity we shall overload the term 'NAT' to mean the composite NAT/firewall.

2.1 STUNT

In [9], the authors propose two approaches for traversing NATs. In the first approach, illustrated in Figure 1(a), both endpoints send an initial SYN with a TTL[4] high enough to cross their own NATs, but small enough that the packets are dropped in the network (once the TTL expires). The endpoints learn the initial TCP sequence number used by their OS's stack by listening for the outbound SYN over PCAP or a RAW socket. Both endpoints inform a globally reachable STUNT server of their respective sequence numbers, following which the STUNT server spoofs a SYNACK to each host with the sequence numbers appropriately set. The ACK completing the TCP handshake goes through the network as usual. This approach has four potential problems. First, it requires the endhost to determine a TTL large enough to cross its own NATs and low enough to not reach the other end's NAT. Such a TTL does not exist when the two outermost NATs share a common interface. Second, the ICMP TTL-exceeded error may be generated in response to the SYN packet and be interpreted by the NAT as a fatal error. Third, the NAT may change the TCP sequence number of the initial SYN such that the spoofed SYNACK based on the original sequence number appears as an out-of-window packet when it arrives at the NAT. Fourth, it requires a third-party to spoof a packet for an arbitrary address, which may be dropped by various ingress and egress filters in the network. These network and NAT issues are summarized in Table 1.

In the second approach proposed in [9], similar to the one proposed in [5], only one endhost sends out a low-TTL SYN packet. This sender then aborts the connection attempt and creates a passive TCP socket on the same address and port. The other endpoint then initiates a regular TCP connection, as illustrated in Figure 1(b). As with the first case, the endhost needs to pick an appropriate TTL value and the NAT must not consider the ICMP error a fatal error. It also requires that the NAT accept an inbound SYN following an outbound SYN – a sequence of packets not normally seen.



(a) STUNT #1 (b) STUNT #2 (c) NATBlaster (d) P2PNAT

Figure 1: Packets generated by various TCP NAT-traversal approaches. Solid lines are TCP/IP and ICMP packets pertaining to the connection attempt while dotted lines are control messages sent over an out-of-band channel.

Approach	NAT/Network Issues	Linux Issues	Windows Issues
STUNT #1	• Determining TTL • ICMP error • TCP Seq# changes • IP Address Spoofing	• Superuser priv.	• Superuser priv. • Setting TTL
STUNT #2	• Determining TTL • ICMP error • SYN-out SYN-in		• Setting TTL
NATBlaster	• Determining TTL • ICMP error • TCP Seq# changes • SYN-out SYNACK-out	• Superuser priv.	• Superuser priv. • Setting TTL • RAW sockets (post WinXP SP2)
P2PNAT	• TCP simultaneous open • Packet flood		• TCP simultaneous open (pre WinXP SP2)
STUNT #1 no-TTL	• RST error • TCP Seq# changes • Spoofing	• Superuser priv.	• Superuser priv. • TCP simultaneous open (pre WinXP SP2)
STUNT #2 no-TTL	• RST error • SYN-out SYN-in		
NATBlaster no-TTL	• RST error • TCP Seq# changes • SYN-out SYNACK-out	• Superuser priv.	• Superuser priv. • RAW sockets (post WinXP SP2) • TCP simultaneous open (pre WinXP SP2)

Table 1: NAT and network issues encountered by various TCP NAT-traversal approaches as well as the implementation issues we encountered.

2.2 NATBlaster

In [3], the authors propose an approach similar to the first STUNT approach but do away with the IP spoofing requirement (Figure 1(c)). Each endpoint sends out a low-TTL SYN and notes the TCP sequence number used by the stack. As before, the SYN packet is dropped in the middle of the network. The two hosts exchange the sequence numbers and each crafts a SYNACK packet the other expects to receive. The crafted packet is injected into the network through a RAW socket; however, this does not constitute spoofing since the source address in the packet matches the address of the endpoint injecting the packet. Once the SYNACKs are received, ACKs are exchanged completing the connection setup. As with the first STUNT approach, this approach requires the endpoint to properly select the TTL value, requires the NAT to ignore the ICMP error and fails if the NAT changes the sequence number of the SYN packet. In addition, it requires that the NAT allow an outbound SYNACK immediately after an outbound SYN – another sequence of packets not normally seen.

2.3 Peer-to-Peer NAT

In [6], the authors take advantage of the simultaneous open scenario defined in the TCP specifications [13]. As illustrated in Figure 1(d), both endpoints initiate a connection by sending SYN packets. If the SYN packets cross in the network, both the endpoint stacks respond with SYNACK packets establishing the connection. If one end's SYN arrives at the other end's NAT and is dropped before that end's SYN leaves that NAT, the first endpoint's stack ends up following TCP simultaneous open while the other stack follows a regular open. In the latter case, the packets on the wire look like the second STUNT approach without the low-TTL and associated ICMP. While [6] does not use port-prediction, the approach can benefit from it when available. As with the second STUNT approach, it requires that the NAT accept an inbound SYN after an outbound SYN. In addition, the approach requires the endhost to retry failed connection attempts in a tight loop until a timeout occurs. If instead of dropping the SYN packet, a NAT responds to it with a TCP RST then this approach devolves into a packet flood until the timeout expires.

2.4 Implementation

We implemented all the above approaches on both Linux and Windows. We also developed a Windows device driver that implements the functionality required by the approaches that are not natively supported by Windows. The first STUNT approach requires superuser privileges under both Windows and Linux to overhear the TCP SYN packet on the wire and learn its sequence number. In order to set the TTL on the first SYN packet, we use the `IP_TTL` socket option under Linux and our driver under Windows. We also implemented the STUNT server and host it behind an ISP that does not perform egress filtering in order to spoof arbitrary addresses. While the server was able to spoof most SYNACKs, it was not successful in spoofing SYNACKs where both the source and destination were in the same administrative domain and the domain used ingress filtering. When possible, we install an additional STUNT server inside such domains. The second STUNT approach requires the driver to set the TTL under windows. The NATBlaster approach requires superuser privileges to learn the sequence number of the SYN and inject the crafted SYNACK through a RAW socket. Due to a restriction introduced in Windows XP SP2, the approach requires the driver to inject this packet. The P2PNAT approach requires the OS to support TCP simultaneous open; this is supported under Linux and Windows XP SP2 but not by Windows XP prior to SP2. On Windows XP SP1 and earlier our driver adds support for this. These implementation issues are summarized in Table 1.

We found that setting the TTL is problematic under Windows; therefore, we consider the consequences of not using it. If the TTL is not reduced, the first SYN sent by one of the hosts reaches the other end's NAT before that end's SYN exits the same NAT. The NAT can either silently drop the inbound packet, or respond with an ICMP unreachable error or a TCP RST/ACK. The response, if any, may trigger transitions in the sender's NAT and OS stack unaccounted for by the approach. If the TTL for the other end's SYN packet is not reduced either, the SYN may reach the intended destination triggering unforeseen transitions. The behavior may be favorable to the ultimate goal if, for instance, it triggers a TCP simultaneous-open, or it may be detrimental if it confuses the stack or NAT. We implement modified versions of the above approaches that do not use low TTLs. The network and implementation issues corresponding to the modified approaches are listed in Table 1.

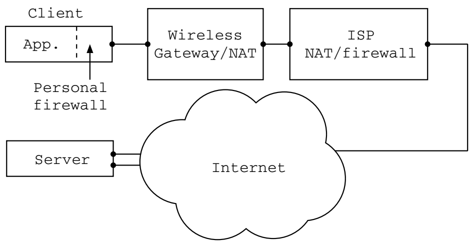


Figure 2: A possible experiment setup for STUNT. Client component is behind multiple NATs and the server component is outside all of them. The behavior determined by STUNT is the composite of the individual NAT behaviors.

Brand	Model	Firmware
3Com	3C857	2.02
Allied Telesyn	AT-AR220E	R1.13
Belkin	F5D5231-4	1.0.0
Buffalo	WYR-G54	1.0 r31
Checkpoint	VPN-1/FireWall-1	(NGAI) release 55
DLink	DI-604	3.30
Linksys	BEFSR41	1.40.2
Linux	iptables	2.4.20
Netgear	RP614	5.13
Netopia	3386	8.0.10
Open BSD	pf	3.5
SMC	SMC7004VBR	R1.00
Trendnet	TW100-S4W1CA	1.02.000.267
USR	8003	1.04 08
VMWare	Workstation	4.5.2
Windows XP	Internet Connection Sharing	SP2

Table 2: NATs tested in the lab, chosen to represent a variety of brands and implementations.

3 Experiment Setup

We have defined the STUNT client-server protocol that both tests NAT/firewall behavior and assists in establishing TCP connections between NAT'ed peers. A complete protocol description is available in [7]. The protocol is implemented by our test applications comprising a client component and server component. As shown in Figure 2, the client is run on a host behind one or more NATs while the server is external to all of them. The STUNT test client detects the composite behavior of all the NATs and firewalls between the client and the server. While both the test client and server require superuser privileges to analyze raw packets on the wire, the requirement can be dropped for the client in exchange for a small loss in functionality. The server, in addition, requires at least two network interfaces to properly differentiate between various NAT port allocation algorithms in use. The tests performed by the client and the NAT characteristics inferred from them are described later in Section 4.

We used the client to test a diverse set of sixteen NATs in

Brand	Sample	Market Survey
Linksys	22.6%	28.8%
D-Link	9.7%	20.2%
Netgear	6.5%	14.7%
Buffalo Technologies	2.2%	10.9%
Belkin	8.6%	4.6%
Other	50.5%	20.9%

Table 3: Observed market share of NAT brands in our sample set and worldwide SOHO/Home WLAN market share of each brand in Q1 2005 according to Synergy Research Group

the lab (Table 2). These include one of each brand of NAT that we could find in online stores in the United States. The NATs tested also include software NAT implementations in popular operating systems. In each lab test the client host was the only host internal to the NAT and the server was on the same ethernet segment as the external interface of that NAT. The client host was set to not generate any network traffic other than that caused by the test client.

In addition to these lab tests, we also tested NAT boxes in the wild. The main reason for this was to expose our test software to a wider range of scenarios than we could reproduce in the lab, thus improving its robustness and increasing our confidence in its operation. In addition, it provided a sample, albeit small, of what types of NAT we can expect to see in practice. For this test, we requested home users to run the test client. This tested 93 home NATs (16 unique brands) being used by CS faculty and students at Cornell and other universities. Test traffic was in addition to typical network activity on the client host and other hosts behind the NAT and included web browsing, instant messaging, peer-to-peer file-sharing, email etc. The resulting data draws from a mix of NAT brands with both new and old models and firmware; however, it admits a bias in the selection of NATs given the relatively small user base with most of them living in the north-eastern United States. This discrepancy is evident in Table 3, where the observed popularity of brands in our sample is listed under ‘Sample’ and the worldwide SOHO/Home WLAN market share of the brands in the first quarter of 2005 as per the Synergy Research Group [18] is listed under ‘Market Survey’. In particular, Buffalo Technologies and Netgear were under-represented in our sample and the percentage of other brands was significantly higher. The full list of home NATs tested is available in [8].

4 NAT TCP Characteristics

In this section, we identify how different NATs affect TCP NAT-traversal approaches. We identify five classifications for NAT behavior; namely, NAT mapping, endpoint packet filtering, filtering response, TCP sequence number preserving, and TCP timers. The classifications and the possible values that a NAT can receive in each class are listed in Table 4. We use the STUNT testing client and server described earlier in Section 3 to classify a collection of sixteen NATs in the lab and ninety-three NATs in the wild. The full set of test results with a wider set of classifications is available in [8].

Classification	Values
Nat Mapping	Independent Address$_\delta$ Port$_\delta$ Address and Port$_\delta$ Connection$_\delta$
Endpoint Filtering	Independent Address Port Address and Port
Response	Drop TCP RST ICMP
TCP Seq#	Preserved Not preserved
Timers	Conservative Aggressive

Table 4: Important categories distinguishing various NATs.

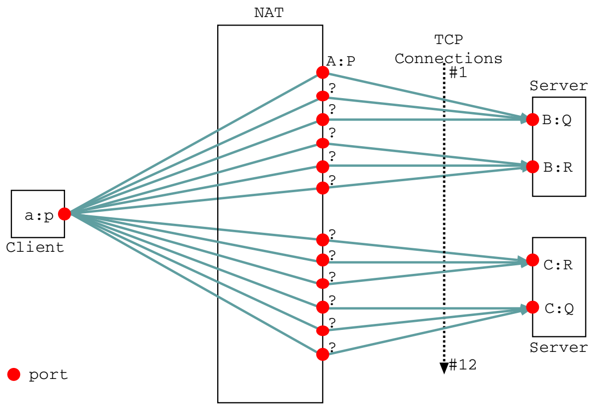


Figure 3: TCP connections established to find NAT Mapping classification. Client uses the same local IP address and port `a:p` to connect three times to two ports `Q` and `R` on two servers at IP addresses `B` and `C`. The pattern of mapped address and port for each connection determines the NAT Mapping classification.

4.1 NAT Mapping

A NAT chooses an external mapping for each TCP connection based on the source and destination IP and port. Some NATs reuse existing mappings under some conditions while others allocate new mappings every time. The *NAT Mapping* classification captures these differences in mapping behavior. This knowledge is useful to endhosts attempting to traverse NATs since it allows them to to predict the mapped address and port of a connection based on previous connections. For UDP, it is known that some NATs

#	From	To	Nat1	Nat2	Nat3	Nat4	Nat5	Nat6
1	a:p	B:Q	A:P	A:P	A:P	A:P$_1$	A:P	A:P
2	a:p	B:Q	A:P	A:P	A:P+1	A:P$_2$	A:P	A:P
3	a:p	B:Q	A:P	A:P	A:P+2	A:P$_3$	A:P	A:P
4	a:p	B:R	A:P	A:P+1	A:P+3	A:P$_4$	A:P+1	A:P
5	a:p	B:R	A:P	A:P+1	A:P+4	A:P$_5$	A:P+1	A:P
6	a:p	B:R	A:P	A:P+1	A:P+5	A:P$_6$	A:P+1	A:P
7	a:p	C:R	A:P	A:P+2	A:P+6	A:P$_7$	A:P+1	A:P+1
8	a:p	C:R	A:P	A:P+2	A:P+7	A:P$_8$	A:P+1	A:P+1
9	a:p	C:R	A:P	A:P+2	A:P+8	A:P$_9$	A:P+1	A:P+1
10	a:p	C:Q	A:P	A:P+3	A:P+9	A:P$_{10}$	A:P	A:P+1
11	a:p	C:Q	A:P	A:P+3	A:P+10	A:P$_{11}$	A:P	A:P+1
12	a:p	C:Q	A:P	A:P+3	A:P+11	A:P$_{12}$	A:P	A:P+1
13	a:s	B:Q	A:S	A:S	A:S	A:S$_1$	A:S	A:S
					⋮			
Classification			NB: Independent	NB:Address and Port$_1$	NB: Connection$_1$	NB: Connection$_\mathfrak{R}$	NB: Port$_1$	NB: Address$_1$

Table 5: NAT Mapping test behavior observed. Nat1–5 show the 5 different mapping patterns that are observed in practice. Nat6 is a possible mapping pattern that has not been observed in our sample set.

assign a fixed address and port for all connections originating from a fixed source address and port [15]. We test for similar behavior for TCP in the STUNT client. The client establishes 12 connections in close succession from a fixed local address and port `a:p` to various server addresses and ports as shown in Figure 3 and tabulated in Table 5. Each connection is closed before the next is initiated. The server echoes back the mapped address and port it perceives to the client. The test is repeated multiple times for different choices of the local port.

We notice several distinct patterns among the mapped ports shown as Nat1–Nat6 in Table 5. Let the mapping allocated for the first connection be called `A:P` for each NAT. Nat1 reuses this mapping as long as the client source address and port of a new connection matches that of the first connection. We classify this behavior as *NB:Independent* since the mapping is determined only by the source address and port and is independent of the destination address and port. This is equivalent to *cone behavior* in [15] extended to include TCP. Nat2 reuses the mapping only if both the source and destination address and port for the new connection match the first connection. Such NATs are classified *NB:Address and Port*$_1$ since both the destination address and port affect the mapping. The subscript '$_1$' signifies that the difference between new mappings, denoted by δ, is 1. [17] shows that for UDP, δ is fixed for many NATs and is usually 1 or 2. We find that the same holds for TCP as well, however, all of the NATs we encountered have $\delta = 1$. Nat3 allocates a new mapping for each new connection, however, each new mapping has port $\delta = 1$ higher than the previous port. We classify Nat3 as *NB:Connection*$_1$. Nat4, like Nat3, allocates a new mapping for each TCP connection but there is no discernable pattern between subsequent mappings. We classify such NATs *NB:Connection*$_\mathfrak{R}$ where the subscript '$_\mathfrak{R}$' indicates a random δ. Nat5 is a variation of Nat2 where the mapping is reused if the destination port matches in addition to the source address and port. Nat6 is similar except the destination address needs to match instead of the port. Together Nat5 and Nat6 are classified *NB:Port*$_1$ and *NB:Address*$_1$ respectively. NATs 2–6 display *symmetric behavior* as per [15].

NAT Mapping	Lab	Wild
NB:Independent	9	70.1%
NB:Address and Port$_1$	3	23.5%
NB:Connection$_1$	3	3.9%
NB:Port$_1$	0	2.1%
NB:Address$_\delta$	0	0.0%
NB:Connection$_\mathfrak{R}$	1	0.5%

Table 6: NAT mapping types observed in a set of 16 NATs in the lab, and that estimated for NATs in the wild based on our sampling of 87 home NATs and worldwide market shares.

Table 6 shows the relative proportion of each type of NAT. Column 2 shows the number of NATs from our testbed of sixteen NATs that were classified as a particular type. A majority of them are NB:Independent. The only one that is NB:Connection$_\mathfrak{R}$ is the NAT implementation in OpenBSD's `pf` utility. We also noticed that our Netgear RP614 NAT with firmware 5.13 is NB:Connection$_1$, however, more recent Netgear NATs such as MR814V2 with firmware 5.3_05 are NB:Independent. Column 3 estimates the behavior of NATs in the wild. The estimates are computed by taking the proportion of each type and brand of NAT from eighty-seven home NATs sampled and scaling them with a correction factor chosen to overcome the bias in our sample. The correction factor for each brand is the ratio between the surveyed and observed market shares presented in Table 3. The factor serves to increase the contribution of under-represented brands in the estimated result and decrease the contribution of over-represented brands.

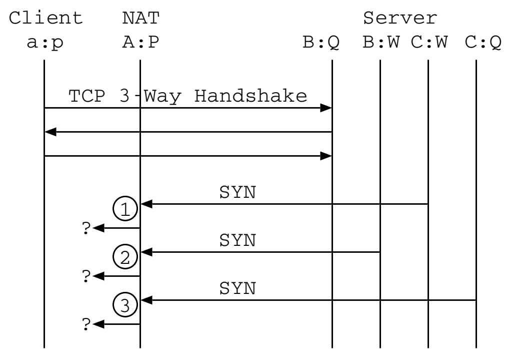


Figure 4: TCP packets exchanged for Endpoint Filtering test. Client establishes a connection to `B:Q`. Packet (1) is an inbound SYN from a different address and port (`C:W`), (2) from the same address but different port (`B:W`) and (3) from the same port but different address (`C:Q`). The response to each of these determines the Endpoint Filtering classification.

While our estimates are indicative of the general trend to the best of our knowledge, we note that in an industry changing at the rate of 47% per year [18] the accuracy of any results is short lived at best. Nevertheless, we estimate a majority of the NATs (70.1%) to be NB:Independent and almost none to be NB:Connection$_{\Re}$. A significant percent (29.9%) of NATs have symmetric behavior. Consequently in a large fraction of cases, multiple connections from the same port will not be assigned the same mapping and applications must employ the more sophisticated port-prediction techniques described later.

4.2 Endpoint Filtering

Both NATs and firewalls may filter inbound packets addressed to a port unless certain conditions are met. If no NAT mapping exists at that port, a NAT is forced to filter the packet since it cannot forward it. If a mapping exists, however, or if it the device is a firewall then it may require that the source address and/or port of the inbound packet match the destination of a preceeding outbound packet. These differences in conditions that trigger filtering is captured by the *Endpoint Filtering* classification. The STUNT test client determines this by first establishing NAT state by connecting to the server. It then requests the server to initiate connections to the mapped address and port from different addresses and ports as shown in Figure 4.

The different filtering behavior observed for the test are tabulated in Table 7. Nat1′ accepts all three SYN packets. Such NATs allow inbound TCP connections independent of the source address and port as long as necessary state exists for routing the request. We classify such NATs as having the endpoint filtering behavior *EF:Independent*. Nat2′ filters all the packets thus requiring the source of the inbound TCP packet match both the address and port of the destination of the connection that created the mapping. The endpoint filtering of such NATs is classified *EF:Address and Port*. Nat3′ and Nat4′ allow inbound packets from the same address or port as the destination address of the connection but filter packets from a different address or port respectively. We classify the endpoint filtering behavior of such NATs as *EF:Address* and *EF:Port* respectively. In general we find that endpoint filtering behavior of a NAT is independent of NAT mapping behavior. The subclassifications of cone NATs defined in [15] translate as follows: *full cone* is equivalent to NB:Independent and EF:Independent, *restricted cone* is NB:Independent and EF:Address and Port and *port restricted cone* is NB:Independent and EF:Port.

Endpoint Filtering	Lab	Wild
Address and Port	12	81.9%
Address	1	12.3%
Independent	3	5.8%

Table 8: NAT endpoint filtering types observed in a set of 16 NATs in the lab, and that estimated for NATs in the wild based on our sampling of 93 home NATs and worldwide market shares.

Sequence	Filtered
SYN-out SYNACK-in	0%
SYN-out SYN-in	13.6%
SYN-out ICMP-in SYNACK-in	6.9%
SYN-out ICMP-in SYN-in	22.4%
SYN-out RST-in SYNACK-in	25.6%
SYN-out RST-in SYN-in	28.1%

Table 9: Percentage of NATs not accepting various packet sequences. Inbound packets (-in) are in response to preceeding outbound packets (-out). ICMP code used is TTL-exceeded (non-fatal error).

Table 8 shows the endpoint filtering classification for sixteen NATs in the lab and the estimated percentage of NATs in the wild. The estimates are computed based on the market survey as described earlier. 81.9% of the NATs are estimated to be EF:Address and Port while only 5.8% are EF:Independent. This implies that in most cases, to establish a connection between two NAT'ed hosts an outbound SYN must be sent from each end before inbound packets are accepted.

4.2.1 TCP State Tracking

NATs implement a state machine to track the TCP stages at the endpoints and determine when connection-state can be garbage-collected. While all NATs handle the TCP 3-way handshake correctly, not all of them implement the corner cases of the TCP state machine correctly thereby prematurely expiring connection-state. The STUNT client and server test how NAT/firewall implementations affect TCP NAT-traversal approaches by replaying the packet sequences observed for these approaches.

#	From	To	Nat1′	Nat2′	Nat3′	Nat4′
1	C:W	A:P	accepted	filtered	filtered	filtered
2	B:W	A:P	accepted	filtered	accepted	filtered
3	C:Q	A:P	accepted	filtered	filtered	accepted
Classification			EF:Independent	EF:Address and Port	EF:Address	EF:Port

Table 7: NAT endpoint filtering behavior observed. Nat1′–Nat4′ show 4 different filtering behaviors that are observed for inbound SYN packets after an internal host establishes a connection from `a:p` to `B:Q` with allocated mapping `A:P`.

Table 9 lists some of the packet sequences tested. We estimate that 13.6% of NATs do not support TCP simultaneous open where an outbound SYN is followed by an inbound SYN. This affects the P2PNAT approach, which requires at least one end support simultaneous open as well as the second STUNT approach. 6.9% filter inbound SYNACK packets after a transient ICMP TTL-exceeded error. A similar number of NATs drop the inbound SYN packet after the ICMP but accept it in the absence of the error. This behavior affects all the approaches that set low-TTLs on SYN packets. A fair number of NATs (28.1%) accept inbound SYN packets even after the SYN packet that created the connection-state is met with a fatal TCP RST. This mitigates the issue of spurious RSTs that some approaches contend with. Not mentioned in the table is the sequence SYN-out SYNACK-out that is required for the NATBlaster approach. We did not test this case widely due to restrictions introduced by Windows XP SP2. In the lab, however, we found that the D-Link NAT (DI-604) does not support it.

4.2.2 Filtering Response

When an inbound packet is filtered by a NAT it can choose to either drop the packet silently or notify the sender. An estimated 91.8% of the NATs simply drop the packet without any notification. The remaining NATs signal an error by sending back a TCP RST acknowledgment for the offending packet.

4.3 Packet Mangling

NATs change the source address and port of outbound packets and the destination address and port of inbound packets. In addition, they need to translate the address and port of encapsulated packets inside ICMP payloads so endhosts can match ICMPs to their respective transport sockets. All the NATs in our sample set either perform the ICMP translation correctly or filter the ICMP packets, which are not always generated in in the first place. Some NATs change the TCP sequence numbers by adding a constant per-flow offset to the sequence number of outbound packets and subtracting the same from the acknowledgment number of inbound packets. We estimate that 8.4% of NATs change the TCP Sequence Number. Consequently in some cases, TCP NAT-traversal approaches that require the initial sequence number of the packet leaving the NAT cannot use the sequence number of the SYN at the end host in its stead.

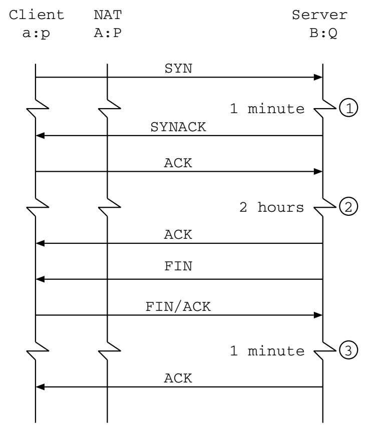


Figure 5: NAT timers in effect during a TCP connection. (1) Timer in SYN_SENT state, (2) Timer in established state, (3) Timer in TIME_WAIT.

4.4 TCP Timers

NATs and firewalls cannot indefinitely hold state since it makes them vulnerable to DoS attacks. Instead they expire idle connections and delete connection-state for crashed or misbehaving endpoints. In addition, they monitor TCP flags and recover state from connections explicitly closed with a FIN/FINACK exchange or RST packets. They might allow some grace time to let in-flight packets and retransmissions be delivered. Once connection-state has been unallocated, any late-arriving packets for that connection are filtered. NATs typically use different timers for these cases as illustrated in Figure 5. At location 1 and 3 in the figure, they use a short timer to expire connections not yet established or connections that have been closed respectively. RFC 1122 [4] requires that endhosts wait for 4 minutes ($2\times$MSL[5]) for in-flight packets to be delivered; however, most operating systems wait for about 1 minute instead. At location 2 in the figure, NATs use a longer timer for idle connections in the established state. The corresponding requirement in RFC 1122 is that TCP stacks should wait for at least 2 hours between sending `TCP-Keepalive` packets over idle connections.

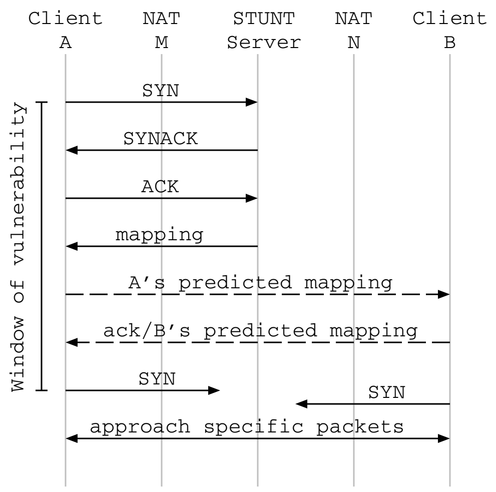


Figure 6: Port-prediction in TCP NAT-Traversal approaches.

The STUNT test client checks the NAT timers for compliance with current practice and RFCs. It does so by performing three timed tests to check each case separately. In the first test, a TCP connections is initiated by the client, but the SYNACK from the server is delayed by a little under a minute. In the second test, the connection is established and left idle for a little under 2 hours, at which point a couple of bytes are sent across from the server. In the third test, the connection is established and then closed but the last ACK from the server is delayed by about a minute. In each case if the packet sent from the server after the introduced delay is delivered to the client then the corresponding timer is termed *conservative*, otherwise it is termed *aggressive*. We estimate only 27.3% of the NATs have conservative timers for all three cases while 35.8% have a conservative timer for the second case. 21.8% of the NATs have an extremely aggressive timer for the second case where they expire an established connection after less than 15 minutes of inactivity. This implies that applications should not rely on idle connections being held open for more than a few minutes.

5 Port Prediction

Port prediction allows a host to predict the mapped address and port for a connection before initiating it. It therefore allows two hosts to initiate a connection with each other's mapped address and port even though the mapping is allocated by the NAT *after* the connection is initiated. Figure 6 shows a typical TCP NAT-traversal attempt using port prediction information. In the figure, we assume that A has already determined the type of NAT it is using. When client A wishes to establish a connection with client B, A first establishes a TCP connection to the STUNT server and learns the mapping. Based on the NB:type of NAT M, A predicts the mapping for the next connection. B does the same and both A and B exchange their predictions over an out-of-bound channel. Each end then initiates a connection to the other's mapped address and port by sending a SYN packet. The remainder of the packets exchanged are designed to reconcile the two TCP attempts into one connection and vary from one NAT-Traversal approach to another as described in Section 2. The period between the first SYN and the SYN for the target connection may be a *window of vulnerability* depending on the type of NAT M. For some types of NAT, if another internal host A′ behind NAT M initiates an outbound connection in this period, M will allocate the mapping predicted by A to the connection from A′ instead.

Port-prediction depends on the NAT Mapping type explored earlier in Section 4.1. If the NAT is of type NB:Independent then the mapping for the connection to the STUNT server will be reused for any connection initiated soon afterward from the same source address and port. Since the reuse of the mapping is completely under the client's control, the window of vulnerability does not exist in this case. However, this approach introduces a latency of $2 \times$RTT[6] to the STUNT server before the mapping can be predicted. For a possible optimization, we noticed that a number of NATs usually allocate a mapped port equal to the source port used by the client. We term these NATs *port preserving*. Clients behind such a NAT can, with high probability, predict the mapped port without first establishing a connection to the STUNT server. If the NAT is not NB:Independent but has a fixed δ then a connection initiated immediately after the server connection will have a mapped port δ higher than the mapped port observed by the server. Since the mapping changes from connection to connection, a "rogue" connection attempt in the window of vulnerability can steal the mapping. In addition, this approach fails if the predicted mapping is already in use, causing the NAT's allocation routine to jump over it onto the next available one.

We implemented port prediction in the STUNT test client and predicted mappings for eighty-three home users for an hour. Every minute, the test client initiates a connection to the STUNT server from a source address and port and learns the mapping allocated. Next it uses the same source address and port to initiate a connection to a remote host setup for the purpose of this experiment. The test client checks the mapping actually observed for the second connection against the one predicted based on the mapping for the first and type of NAT. Port-prediction is successful if and only if they match. The predictions are performed while users use their host and network normally. This includes web browsers, email readers, instant messaging and file-sharing applications running on the client host and other hosts behind the same NAT. 88.9% of the NB:Independent NATs are port preserving. This represents a big win for interactive applications that implement the optimization above. We find that in 81.9% of the cases the

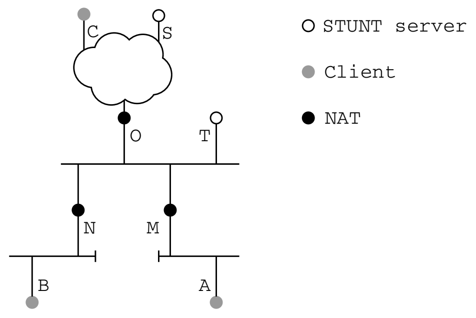


Figure 7: Problematic scenarios for port prediction.

port was predicted correctly every time. This includes all but one of the NB:Independent NATs and 37.5% of the non-NB:Independent NATs. For the remaining 62.5% of the latter variety, at least one time out of the sixty another host or application stole the mapping the test client predicted for itself. In one particular case, the client host behind a NB:Connection$_1$ NAT was infected with a virus that generated several randomly-addressed SYN packets per second causing all predictions to fail! In another case, the user initiated a VPN connection midway through the test causing all subsequent requests to be sent over the VPN and thus through a different type of NAT. This suggests that long-running applications may cache the NAT mapping type for some time but must revalidate it from time to time. Overall, in 94.0% of the cases, more than three-fourths of the port predictions were correct. Hence after a failed attempt if an application simply retries the connection, it is likely to succeed.

5.1 Problems

Port prediction has several corner cases where it can fail. In Figure 7 if `A` uses STUNT server `T` to predict an address and port when trying to establish a connection to `C` it would end up learning NAT `M`'s external address instead of NAT `O`'s external address. Port prediction requires that the line of NATs between the client and STUNT server be the same as that between the client and the most external NAT of the endpoint it wishes to connect with. Therefore `A` somehow needs to discover `S` in order to connect to `C`. If, however, `A` wishes to communicate with `B` and both use STUNT server `S` then their port prediction attempts can interfere with each other, preventing either from correctly predicting the port. In addition, even if the port is predicted correctly, both `A` and `B` will end up using `O`'s external address. This scenario is called a *hairpin translation* since `A`'s SYN addressed to `B`'s predicted address and port (`O`'s external address in this case) will be delivered to `O`, which needs to send it back out on an *internal* interface. Not all NATs handle hairpin translations correctly and we estimate this erroneous behavior to be as high as 72.8% based on tests performed by the STUNT test client.

The port-prediction technique described earlier does not handle NB:Connection$_{\mathfrak{R}}$ NATs since sequential connections are randomly assigned. [3] proposes an interesting technique for handling such cases that uses the birthday paradox to cut down on the number of guesses before a collision is found. The technique initiates 439 connections such that the guessed port will match one of them with 95% probability. Unfortunately, we find that some NATs, like Netgear, limit the total number of pending connection attempts to 1000 causing this approach to quickly stifle the NAT. Fortunately, very few NATs demonstrate NB:Connection$_{\mathfrak{R}}$ behavior mitigating the problem.

6 TCP Establishment

In this section we estimate the success of the various NAT traversal approaches as well as report our experience with peer-to-peer TCP establishment for a small wide-area testbed. The success of TCP NAT-traversal approaches depends on the behavior of all NATs between the two end-hosts as well as the activity of other hosts behind the NATs. Section 4 analyzes a variety of NATs in isolation while Section 5 analyzes competing network activity and its effect on port prediction. Combining the results from these sections we can quantitatively estimate the success of each NAT traversal approach.

We make the following assumptions about the deployment of the TCP-traversal approaches. We assume that STUNT servers are deployed widely enough to ensure that for each pair of hosts, there is a STUNT server that meets the port-prediction requirements and can spoof packets appearing to come from the mapped address and port of each host. We assume that endhost stacks can be fixed so all software issues at the ends are resolved. Lastly, since we lack data to model the scenarios presented in Section 5.1 we assume the contribution from such scenarios to be negligible. As a result of these assumptions, our estimates may be optimistic.

Peer-to-peer TCP establishment depends on the NATs at both ends. An endpoint with an unpredictable NAT may still be able to establish a connection if the other endpoint's NAT is predictable but not if it is unpredictable. We estimate TCP connectivity in the wild by considering all pairs of NAT behavior observed in practice. Figure 8 plots the estimated success rate of various TCP NAT traversal approaches. We plot the two STUNT approaches (#1 and #2), NATBlaster and P2PNAT as proposed in [9, 3, 6]. In addition, we plot modified versions of STUNT #1 and #2 and NATBlaster approaches that do not use low-TTLs. We also plot a modified version of the P2PNAT approach that uses port-prediction. There is a race condition between the SYN packets in some of these approaches that leads to spurious packets for certain NAT-pairs. The light-gray bars repre-

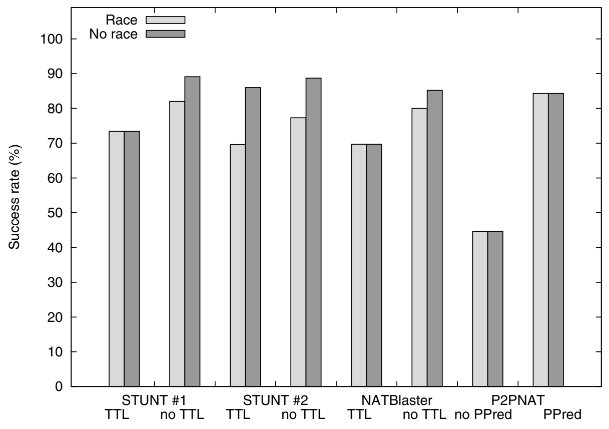


Figure 8: Estimated peer-to-peer TCP NAT traversal success rates of various approaches. With prevailing race conditions (Race) the success rate is lower than when races are resolved for the best outcome (No race).

sent the success rate when each end has an equal chance of winning the race; this corresponds to simultaneous invocation of the approach on both ends. The dark-gray bar represents the success rate when the race is broken in favor of a successful connection; this corresponds to two separate invocations of the approach where for the first invocation, one end is initiated slightly before the other while for the second invocation, the order is reversed. The attempt is declared successful if either invocation succeeds in establishing a TCP connection.

As shown in the graph, the original approaches proposed succeed 44.6% and 73.4% of the time for P2PNAT and STUNT #1 respectively. Breaking the race condition in the original STUNT #2 approach by trying it once from each end boosts its success to 86.0%. Similarly, adding port prediction to the P2PNAT approach allows it to handle symmetric NATs increasing its success rate to 84.3%. Surprisingly, modifying the original approaches to not use low-TTLs benefits all of them by ∼5%! Breaking the race conditions thus introduced yields the best success rates of 89.1% and 88.7% for the two modified STUNT approaches and 85.2% for the modified NATBlaster approach.

The unexpected benefits to *not* using low-TTL SYNs are explained as follows. A large fraction of NATs silently drop the first SYN packet (Section 4.2.2) and only a small fraction of NATs filter inbound SYN packets after the outbound SYN packet (Table 9). Consequently in a large number of cases, the modified approaches end up triggering TCP simultaneous open even though they do not intend to. The small penalty they pay for NATs that generate a TCP RST response is more than compensated for by the successful TCP simultaneous opens. This advantage is eroded away if more NATs respond with TCP RST packets or if the endhost's operating system does not support TCP simultaneous open.

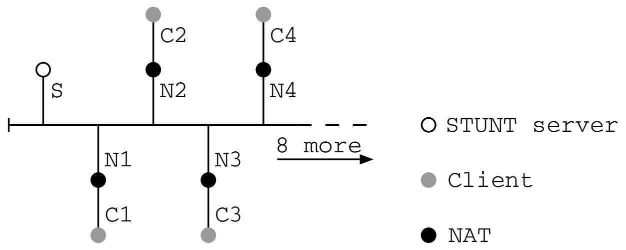


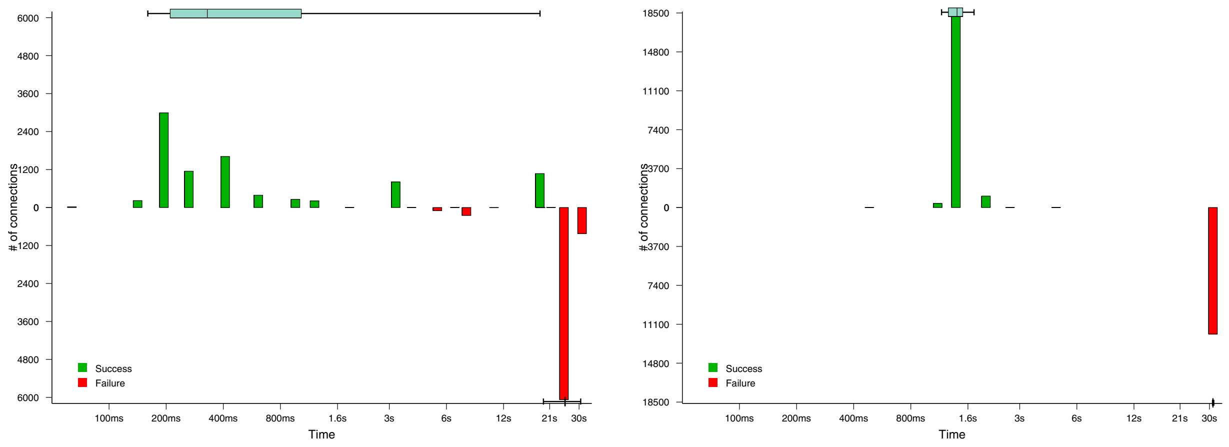
Figure 9: Network of 12 clients for peer-to-peer TCP NAT-traversal tests.

6.1 Implementation

We implemented the above approaches in a peer-to-peer program written in C. The program was run on 12 windows clients connected in a LAN as shown in Figure 9 as well as a slightly larger set of 20 clients connected over a WAN. Each client randomly picks another client and attempts to establish TCP with it. While all the approaches work as advertised, we limit this discussion to STUNT #2 without low-TTL and P2PNAT with port prediction. This is because we find that a large global deployment of STUNT #1 and NATBlaster approaches is impractical; the STUNT #1 approach requires a broad deployment of servers that can spoof arbitrary packets while the NATBlaster approach requires RAW socket functionality that has been removed following security concerns in Windows XP.

Figure 10 shows a semi-log plot of the time taken by each of the approaches to establish a connection or report a failure in a low-latency network. The time-distribution of successful connections (plotted above the x-axis) varies largely for the P2PNAT approach while that of the second STUNT approach is very consistent. This is because the P2PNAT approach repeatedly initiates a connection until one of them succeeds or a timeout occurs while the second STUNT approach only initiates one connection. From the graph in Figure 10(a), a fair number of connections do not succeed until 21 seconds into the connection attempt, thus requiring a large timeout to achieve the estimated success rate determined earlier. In several cases a dangerous side-effect of such a large timeout is observed when port-prediction fails and a peer's NAT responds with TCP RST packets. This causes the P2PNAT approach to generate a SYN-flood as the endhost repeatedly retries the connection until the timeout expires. Admittedly, this problem does not exist in the original P2PNAT approach since it does not advocate port-prediction.

Overall, we find that TCP based protocols can traverse NATs most of the time. Applications must perform port prediction to deal with "delta" type NATs and employ out-of-band signalling to setup connections while satisfying se-



(a) P2PNAT (b) STUNT #2

Figure 10: Semi-log plot of time taken to successfully establish a connection or report a failure.

curity contraints at the NATs. Non-latency-sensitive applications must also attempt a couple of connections from either side before giving up. Furthermore, applications must not rely on NATs preserving idle connections for more than a few minutes and should not expect TCP sequence numbers to remain unchanged end-to-end. While all the TCP NAT traversal approaches falter under certain scenarios, the second STUNT approach is the most robust in establishing peer-to-peer TCP connections in the Internet today. It succeeds 88% of the time and does not require spoofing, RAW sockets, or superuser privileges. It works with legacy TCP stacks that do not support TCP simultaneous open, while taking advantage of the same in modern operating systems to provide 100% success in many common scenarios. Finally, we find that it is the simplest to implement out of all the approaches and works under both Linux and Windows. We encourage application developers to adapt this approach to provide TCP NAT traversal in their peer-to-peer applications that currently cannot establish TCP between NAT'ed peers.

7 Related Work

NAT traversal is an idea that has long existed since it was first proposed for UDP by Dan Kegel, and used for P2P gaming, in the late 90's. His basic approach was standardized and published in [15]. The UDP approach has resulted in several working documents and Internet drafts that classify UDP behavior of NATs [11] and propose standardization of the same [1]. The area of TCP NAT-traversal without explicit control of the NAT, however, is fairly new. Several approaches have been analyzed in this paper [9, 5, 3, 6] and have been demonstrated to traverse NATs in the current internet. [6] includes a similar study where the authors test a small subset of UDP and TCP NAT characteristics for a number of NATs. We present the first broad study of NAT behavior and peer-to-peer TCP establishment for a comprehensive set of TCP NAT traversal techniques over a wide range of commercial NAT products.

Protocols such as UPnP [12] and MIDCOM [16] allow endhost applications to explicitly control the NAT in order to facilitate peer-to-peer connections. The downside of this type of approach is that they require that these protocols exist and are enabled in the NAT. An application developer cannot depend on either of these, and so for the time being they are not an attractive option.

Another endhost approach to TCP connectivity includes TURN [14], where TCP data is proxied by a third party. This third party represents a potential network bottleneck. Teredo [10] allows IPv6 packets to traverse IPv4 NATs by tunneling IPv6 over UDP. Here, TCP runs natively in the sense that it is layered directly above IPv6. Teredo has been implemented in the Windows OS.

8 Conclusion and Future Work

This paper presents the first measurement study of a comprehensive set of NAT characteristics as they pertain to TCP. While this study shows that there are a significant number of problems with TCP traversal of current NATs, it nevertheless gives us much to be optimistic about. Even with existing NATs, which have not been designed with TCP NAT traversal in mind, we are able to show a 88% average success rate for TCP connection establishment with NATs in the wild, and a 100% success rate for pairs of certain common types of NAT boxes. These numbers are especially encouraging given that only a couple years ago, it was widely assumed that TCP NAT traversal was simply not possible. While a failure rate of 11% is not acceptable for many applications, the users of those applications at least have the option of buying NAT boxes that allow TCP traversal, and NAT vendors have the option of designing their NAT boxes to be more TCP friendly.

This study is limited in several respects. First, we did not test all existing NAT boxes. For the most part our study was limited to NAT boxes available in North America. Second, our field test is too small and biased to be able to accurately predict success rates broadly. Using market data helps, but even this data was limited in scope and in detail. Third, we tested only for home NATs. Port prediction in enterprise networks for "delta" type NATs will succeed less often, and it would be useful to measure this. Fourth, although TCP NAT traversal techniques apply broadly to firewalls, we did not test firewalls outside the context of NAT. Nor did we test IPv4-IPv6 translation gateways. Finally, like most measurement studies, this is a snapshot in time. Indeed, during the course of this project, new versions of the same NAT product exhibited different behavior from previous versions. Moving forwards, we hope that our TCP NAT traversal test suite can continue to be used to broaden our knowledge of NAT and firewall characteristics as well as to track trends in NAT products and their deployment.

Acknowledgments

The authors wish to thank Bryan Ford, Andrew Biggadike, and the anonymous IMC reviewers for valuable feedback on early drafts of this paper. We also wish to thank Synergy Research Group, Inc., for providing up-to-date market survey data. Finally, we wish to thank the many volunteers who ran the STUNT test client and the peer-to-peer TCP tests on their systems and submitted the results.

References

[1] AUDET, F., AND JENNINGS, C. Internet draft: Nat behavioral requirements for unicast udp, July 2005. Work in progress.

[2] BASET, S. A., AND SCHULZRINNE, H. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. Tech. Rep. CUCS-039-04, Columbia University, New York, NY, Sept. 2004.

[3] BIGGADIKE, A., FERULLO, D., WILSON, G., AND PERRIG, A. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *Proceedings of ACM SIGCOMM ASIA Workshop* (Beijing, China, Apr. 2005).

[4] BRADEN, R. RFC 1122: Requirements for Internet Hosts – Communication Layers, Oct. 1989.

[5] EPPINGER, J. L. TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem. Tech. Rep. CMU-ISRI-05-104, Carnegie Mellon University, Pittsburgh, PA, Jan. 2005.

[6] FORD, B., SRISURESH, P., AND KEGEL, D. Peer-to-peer communication across network address translators. In *Proceedings of the 2005 USENIX Annual Technical Conference* (Anaheim, CA, Apr. 2005).

[7] GUHA, S. STUNT - Simple Traversal of UDP Through NATs and TCP too. Work in progress. [online] http://nutss.gforge.cis.cornell.edu/pub/draft-guha-STUNT-00.txt.

[8] GUHA, S. STUNT Test Results. [online] http://nutss.gforge.cis.cornell.edu/stunt-results.php.

[9] GUHA, S., TAKEDA, Y., AND FRANCIS, P. NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity. In *Proceedings of SIGCOMM'04 Workshops* (Portland, OR, Aug. 2004), pp. 43–48.

[10] HUITEMA, C. Internet draft: Teredo: Tunneling ipv6 over udp through nats, Apr. 2005. Work in progress.

[11] JENNINGS, C. Internet draft: Nat classification test results, Feb. 2005. Work in progress.

[12] MICROSOFT CORPORATION. UPnP – Universal Plug and Play Internet Gateway Device v1.01, Nov. 2001.

[13] POSTEL, J. RFC 761: DoD standard Transmission Control Protocol, Jan. 1980.

[14] ROSENBERG, J., MAHY, R., AND HUITEMA, C. Internet draft: TURN – Traversal Using Relay NAT, Feb. 2004. Work in progress.

[15] ROSENBERG, J., WEINBERGER, J., HUITEMA, C., AND MAHY, R. RFC 3489: STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), Mar. 2003.

[16] STIEMERLING, M., QUITTEK, J., AND TAYLOR, T. MIDCOM Protocol Semantics, June 2004. Work in progress.

[17] TAKEDA, Y. Internet draft: Symmetric NAT Traversal using STUN, June 2003. Work in progress.

[18] VANCE, A., Ed. *Q1 2005 Worldwide WLAN Market Shares*. Synergy Research Group, Inc., Scottsdale, AZ, May 2005, p. 40.

Notes

[1] Network behind a NAT or fire wall

[2] Throughout this paper, the term NAT is understood to include firewalls.

[3] sometimes referred to as *dual* or *double NAT*

[4] IP time-to-live field

[5] Maximum Segment Length

[6] Round-trip time

Predicting short-transfer latency from TCP arcana: A trace-based validation

Martin Arlitt
HP Labs/University of Calgary
Palo Alto, CA 94304
Martin.Arlitt@hp.com

Balachander Krishnamurthy
AT&T Labs–Research
Florham Park, NJ 07932
bala@research.att.com

Jeffrey C. Mogul
HP Labs
Palo Alto, CA 94304
Jeff.Mogul@hp.com

Abstract

In some contexts it may be useful to predict the latency for short TCP transfers. For example, a Web server could automatically tailor its content depending on the network path to each client, or an "opportunistic networking" application could improve its scheduling of data transfers.

Several techniques have been proposed to predict the latency of short TCP transfers based on online measurements of characteristics of the current TCP connection, or of recent connections from the same client. We analyze the predictive abilities of these techniques using traces from a variety of Web servers, and show that they can achieve useful accuracy in many, but not all, cases. We also show that a previously-described model for predicting short-transfer TCP latency can be improved with a simple modification. Ours is the first trace-based analysis that evaluates these prediction techniques across diverse user communities.

1 Introduction

It is often useful to predict the latency (i.e., duration) of a short TCP transfer before deciding when or whether to initiate it. Network bandwidths, round-trip times (RTTs), and loss rates vary over many orders of magnitude, and so the transfer latency for a given data item can vary similarly.

Examples where such predictions might be useful include:

- a Web server could automatically select between "low-bandwidth" and "high-bandwidth" versions of content, with the aim of keeping the user's download latency below a threshold [9, 16].
- A Web server using shortest-remaining-processing-time (SRPT) scheduling [19] could better predict overall response times if it can predict network transfer latency, which in many cases is the primary contributor to response time.
- An application using *opportunistic networking* [20] might choose to schedule which data to send based on an estimate of the duration of a transfer opportunity and predictions of which data items can make the most effective use of that opportunity.

There are several possible ways to define "short" TCP transfers. Models for TCP performance typically distinguish between long flows which have achieved steady state, and short flows which do not last long enough to leave the initial slow-start phase. Alternatively, one could define short in terms of an arbitrary threshold on transfer length. While defining "short" in terms of slow-start behavior is less arbitrary, it is also less predictable (because the duration of slow-start depends on unpredictable factors such as cross traffic and packet loss), and so for this paper we use a definition based on transfer length. Similarly, while transfer length could be defined in terms of the number of data packets sent, this also depends on unpredictable factors such as MTU discovery and the interactions between application buffering and socket-level buffering. So, for simplicity, in this paper we define "short" in terms of the number of bytes transferred.

Several techniques have previously been proposed for automated prediction of the transfer time for a short TCP transfer. Some of these techniques glean their input parameters from characteristics of TCP connections, such as round-trip time (RTT) or congestion window size (cwnd), that are not normally exposed to the server application. We call these characteristics *TCP arcana*. These characteristics can then be used in a previously-described model for predicting short-transfer latency [2]. Other techniques use observations of the actual latency for past transfers to the same client (or to similarly located clients), and assume that past performance is a good predictor of future performance.

In this paper, we use packet-level traces captured near a variety of real Web servers to evaluate the ability of techniques based on both TCP arcana and historical observation to predict short transfer latencies. We show that the previously-described model does not quite fit the observations, but that a simple modification to the model greatly improves the fit. We also describe an experiment suggesting (based on a limited data set) that RTT observations could be used to discriminate, with modest accuracy, between dialup and non-dialup paths.

This work complements previous work on predicting the *throughput* obtained by *long* TCP transfers. He *et al.* [7] characterized these techniques as either formula-based or history-based; our TCP arcana approach is formula-based.

2 Latency prediction techniques

We start with the assumption that an application wishing to predict the latency of a short transfer must do so as early as possible, before any data has been transferred. We also assume that prediction is being done at the server end of a connection that was initiated by a client; although the approaches could be extended to client-side prediction, we have no data to evaluate that scenario.

We examine two prediction approaches in this paper:

- The **initial-RTT** approach: The server's first possible measurement of the connection RTT is provided by the interval between its initial SYN|ACK packet and the client's subsequent ACK. For short transfers, this RTT measurement is often sufficient to predict subsequent data transfer latency to this client. This approach was first proposed by Mogul and Brakmo [15] and discussed in [16]. We describe it further in Section 2.1.
- The **recent-transfers** approach: A server can predict the data transfer bandwidth to a given request based on recently measured transfer bandwidths to the same client. This approach, in the context of Web servers, was proposed in [9]; we describe it further in Section 2.2.

2.1 Prediction from initial RTTs

Suppose one wants to predict the transfer latency, for a response of a given length over a specific HTTP connection, with no prior information about the client and the network path, and before having to make the very first decision about what content to send to the client. Let us assume that we do not want the server to generate extra network traffic or cause extra delays. What information could one glean from the TCP connection before it is too late?

Figure 1 shows a timeline for the packets sent over a typical non-persistent HTTP connection. (We assume that the client TCP implementation does not allow the client application to send data until after the 3-way handshake; this is true of most common stacks.) In this timeline, the server has to make its decision immediately after seeing the GET-bearing packet (P_4) from the client.

It might be possible to infer network path characteristics from the relative timing of the client's first ACK-only (P_3) and GET (P_4) packets, using a packet-pair method [11]. However, the initial-RTT predictor instead uses the path's RTT, as measured between the server's SYN|ACK packet (P_2) and the client's subsequent ACK-only packet (P_3). Since these two packets are both near-minimum length, they provide a direct measurement of RTT, in the absence of packet loss.

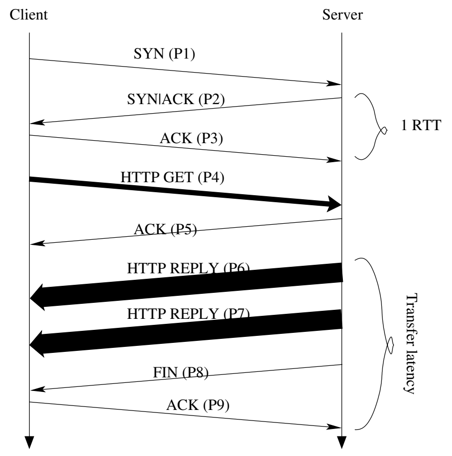


Figure 1: Timeline: typical HTTP connection

Why might this RTT be a useful predictor of transfer latency?

- Many last-hop network technologies impose both high delay and low bandwidth. For example, dialup modems almost always add about 100ms to the RTT [4, 5] and usually limit bandwidth to under 56Kb/s. If we observe an RTT much lower than 100ms, we can infer that the path does not involve a modem. (See Section 5 for quantitative evidence.) A similar inference might be made about some (perhaps not all) popular low-bandwidth wireless media.
- Even when the end-to-end bandwidth is large, the total transfer time for short responses depends mostly on the RTT. (Therefore, an HTTP request header indicating client connection speed would not reliably predict latency for such transfers.)

Cardwell *et al.* [2] showed that for transfers smaller than the limiting window size, the expected latency to transfer d segments via TCP, when there are no packet losses, is *approximated* by

$$E[latency] = RTT \cdot log_{\gamma}(\frac{d(\gamma - 1)}{w_1} + 1) \qquad (1)$$

where

- γ depends on the client's delayed-ACK policy; reasonable values are 1.5 or 2 (see [2] for details).
- w_1 depends on the server's initial value for `cwnd`; reasonable values are 2, 3, or 4 (see [2] for details).
- $d = \lceil \frac{len}{MSS} \rceil$
- len is the number of bytes sent.
- MSS is the TCP maximum segment size for the connection.

Note that median Web response sizes (we use the definition of "response" from the HTTP specification [6]) are typically smaller than the limiting window size; see Section 3.4.

End-to-end bandwidth limits and packet losses can only increase this latency. In other words, if we know the RTT and response size, then we can predict a lower bound for the transfer latency.

We would like to use the RTT to predict the transfer latency as soon as possible. Therefore, the first time a server sees a request from a given client, it has only one RTT measurement to use for this purpose. But if the client returns again, which RTT measurement should the server use for its prediction? It could use the most recent measurement (that is, from the current connection), as this is the freshest; it could use the mean of all measurements, to deal with noise; it could use an exponentially smoothed mean, to reduce noise while favoring fresh values; it could use the minimum measurement, to account for variable queueing delays; or it could use the maximum measurement, to be conservative.

"Most recent," which requires no per-client state, is the simplest to implement, and this is the only variant we have evaluated.

2.2 Prediction from previous transfers

Krishnamurthy and Wills originally described the notion of using measurements from previous transfers to estimate the connectivity of clients [9]. A prime motivation of this work was to retain poorly connected clients, who might avoid a Web site if its pages take too long to download. Better connected clients could be presented enhanced versions of the pages.

This approach is largely passive: it examines server logs to measure the inter-arrival time between base-object (HTML) requests and the requests for the first and last embedded objects, typically images. Exponentially smoothed means of these measurements are then used to classify clients. A network-aware clustering scheme [8] was used as an initial classification mechanism, if a client had not been seen before but another client from the same cluster had already used the site. Krishnamurthy and Wills used a diverse collection of server logs from multiple sites to evaluate the design, and Krishnamurthy *et al.* presented an implementation [10], using a modified version of the Apache server, to test the impact of various server actions on clients with different connectivity.

The recent-transfers approach that we study in this paper is a simplification of the Krishnamurthy and Wills design. Because their measurements use Web server logs, this gave them enough information about page structure to investigate the algorithm's ability to predict the download time for an entire page, including embedded objects. We have not extracted object-relationship information from our packet traces, so we only evaluated per-response latency, rather than per-page latency. On the other hand, most server logs provide timing information with one-second resolution, which means that a log-based evaluation cannot provide the fine-grained timing resolution that we got from our packet traces.

2.3 Defining transfer latency

We have so far been vague about defining "transfer latency." One might define this as the time between the departure of the first response byte from the server and the arrival of the last response byte at the client. However, without perfect clock synchronization and packet traces made at every host involved, this duration is impossible to measure.

For this paper, we define transfer latency as the time between the departure of the first response byte from the server and the arrival *at the server* of the acknowledgment of the last response byte. (Figure 1 depicts this interval for the case of a non-persistent connection.) This tends to inflate our latency measurement by approximately RTT/2, but because path delays can be asymmetric we do not attempt to correct for that inflation. We are effectively measuring an upper bound on the transfer latency.

3 Methodology

We followed this overall methodology:

- **Step 1:** collect packet traces near a variety of Web servers with different and diverse user populations.
- **Step 2:** extract the necessary connection parameters, including client IDs, from these raw traces to create intermediate traces.
- **Step 3:** evaluate the predictors using simple simulator(s) driven from the intermediate traces.

Although the prediction mechanisms analyzed in this paper are not necessarily specific to Web traffic, we limited our trace-based study to Web traffic because we have not obtained significant and diverse traces of other short-transfer traffic. It might be useful to capture traffic near busy e-mail servers to get another relevant data set, since e-mail transfers also tend to be short [13].

Given that we are defining "short" TCP transfers in terms of the number of data bytes sent, we analyzed three plausible thresholds: 8K bytes, 16K bytes, and 32K bytes; this paper focuses on the 32K byte threshold. (The response-size distributions in Figure 2 support this choice.)

3.1 Trace sets

We collected trace sets from several different environments, all in North America. For reasons of confidentiality, we identify these sets using short names:

- **C2:** Collected on a corporate network
- **U2,U3,U4:** Collected at a University
- **R2:** Collected at a corporate research lab

In all cases, the traces were collected on the public Internet (not on an Intranet) and were collected relatively near exactly one publicly-accessible Web server.

We collected full-packet traces, using tcpdump, and limited the traces to include only TCP connections to or from the local Web server.

While we wanted to collect traces covering an entire week at each site, storage limits and other restrictions meant that we had to collect a series of shorter traces. In order to cover representative periods over the course of a week (May 3–9, 2004), we chose to gather traces for two to four hours each day: 9:00AM-11:00AM Monday, Wednesday, and Friday; 2:00PM-4:00PM Tuesday and Thursday; and 10:00AM-2:00PM Saturday and Sunday (all are local times with respect to the trace site: MST for C2, MDT for U2, and PDT for R2). We additionally gathered two 24-hour (midnight to midnight) traces at the University: U3 on Thursday, Aug. 26, 2004, and U4 on Tuesday, Aug. 31, 2004.

3.2 Are these traces representative?

We certainly would prefer to have traces from a diverse sample of servers, clients, and network paths, but this is not necessary to validate our approach. Our goal is not to predict the latencies seen by all client-server pairs in the Internet, but to find a method for a given server to predict the latencies that it itself (and only itself) will encounter in the near future.

It is true that some servers or client populations might differ so much from the ones in our traces that our results do not apply. Although logistical and privacy constraints prevent us from exploring a wider set of traces, our analysis tools are available at *http://bro-ids.org/bro-contrib/network-analysis/akm-imc05/* so that others can test our analyses on their own traces.

The results in Section 4.6 imply that our equation-based predictor works well for some sites and not so well for others. One could use our trace-based methodology to discover if a server's response latencies are sufficiently predictable before deciding to implement prediction-based adaptation at that server.

3.3 Trace analysis tools

We start by processing the raw (full-packet binary) traces to generate one tuple per HTTP request/response exchange. Rather than write a new program to process the raw traces, we took advantage of *Bro*, a powerful tool originally meant for network intrusion detection [17]. Bro includes a *policy script interpreter* for scripts written in Bro's custom scripting language, which allowed us to do this processing with a relatively simple policy script – about 800 lines, including comments. We currently use version 0.8a74 of Bro.

Bro reduces the network stream into a series of higher level events. Our policy script defines handlers for the relevant events. We identify four analysis states for a TCP connection: **not_established, timing_SYN_ACK, established,** and **error_has_occurred**. We also use four analysis states for each HTTP transaction: **waiting_for_reply, waiting_for_end_of_reply, waiting_for_ack_of_reply,** and **transaction_complete**. (Our script follows existing Bro practice of using the term "reply" in lieu of "response" for state names.)

Progression through these states occurs as follows. When the client's SYN packet is received, a data structure is created to retain information on the connection, which starts in the **not_established** state. When the corresponding SYN|ACK packet is received from the server, the modeled connection enters the **timing_SYN_ACK** state, and then to the **established** state when the client acknowledges the SYN|ACK.

We then wait for **http_request()** events to occur on that connection. When a request is received, a data structure is created to retain information on that HTTP transaction, which starts in the **waiting_for_reply** transaction state. On an **http_reply()** event, that state becomes **waiting_for_end_of_reply**. Once the server has finished sending the response, the transaction state is set to **waiting_for_ack_of_reply**. Once the entire HTTP response has been acknowledged by the client, that state is set to **transaction_complete**. This design allows our script to properly handle persistent and pipelined HTTP connections.

Our analysis uses an additional state, **error_has_occurred**, which is used, for example, when a TCP connection is reset, or when a packet is missing, causing a gap in the TCP data. All subsequent packets on a connection in an **error_has_occurred** state are ignored, although RTT and bandwidth estimates are still recorded for all HTTP transactions that completed on the connection before the error occurred.

For each successfully completed and successfully traced HTTP request/response exchange, the script generates one tuple that includes the timestamp of the arrival time of the client's acknowledgement of all outstanding response data; the client's IP address; the response's length, content-type, and status code; the position of the response in a persistent connection (if any); and estimates of the initial RTT, the MSS, the response transfer latency, and the response transfer bandwidth. The latency is estimated as described in Section 2.3, and the bandwidth estimate is then computed from the latency estimate and the length.

These tuples form an intermediate trace, convenient for further analysis and several orders of magnitude smaller than the original raw packet trace. For almost all of our subsequent analysis, we examine *only* responses with status code $= 200$, since these are the only ones that should always carry full-length bodies.

3.3.1 Proxies and robots

Most Web servers receive requests from multi-client proxy servers, and from robots such as search-engine crawlers; both kinds of clients tend to make more frequent requests than single-human clients. Requests from proxies and robots skew the reference stream to make the average connection's bandwidth more predictable, which could bias our results in favor of our prediction mechanisms.

We therefore "pruned" our traces to remove apparent proxies and robots (identified using a separate Bro script); we then analyzed both the pruned and unpruned traces.

In order to avoid tedious, error-prone, and privacy-disrupting techniques for distinguishing robots and proxies, we tested a few heuristics to automatically detect such clients:

- Any HTTP request including a `Via` header probably comes from a proxy. The converse is not true; some proxies do not insert `Via` headers.
- Any request including a `From` header probably comes from a robot. Not all robots insert `From` headers.
- If a given client IP address generates requests with several different `User-Agent` headers during a short interval, it is probably a proxy server with multiple clients that use more than one browser. It could also be a dynamic IP address that has been reassigned to a different client, so the time scale affects the accuracy of this heuristic. We ignore `User-Agent: contype` headers, since this is an artifact of a particular browser [12, 14].

The results of these tests revealed that the `From` header is not widely used, but it is a reasonable method for identifying robots in our traces. Our test results also indicated that simply excluding all clients that issued a `Via` or `User-Agent` header would result in excessive pruning.

An analysis of the `Via` headers suggested that components such as personal firewalls also add this header to HTTP requests. As a result, we decided to only prune clients that include a `Via` header that can be automatically identified as a multi-client proxy: for example, those added by a Squid, NetApp NetCache, or Inktomi Traffic-Server proxy.

We adopted a similar approach for pruning clients that sent multiple different `User-Agent` headers. First, we require that the `User-Agent` headers be from well-known browsers (e.g., IE or Mozilla). These browsers typically form the `User-Agent` header in a very structured format. If we cannot identify the type of browser, the browser version, and the client OS, we do not use the header in the analysis. If we then see requests from two different browsers, browser versions, or client OSs coming from the same IP address in the limited duration of the trace, we consider this to be a proxy, and exclude that client from the prune trace.

We opted to err (slightly) on the side of excessive pruning, rather than striving for accuracy, in order to reduce the chances of biasing our results in favor of our predictors.

3.4 Overall trace characteristics

Table 1 shows various aggregate statistics for each trace set, to provide some context for the rest of the results. For reasons of space, we omit day-by-day statistics for C2, R2, and U2; these show the usual daily variations in load, although C2 and R2 peak on the weekend, while U2 peaks during the work week. The table also shows totals for the pruned versions of each trace set. Finally, the table shows total response bytes, response count, and mean response size for just the status-200 responses on which most subsequent analyses are based.

We add "p" to the names of trace sets that have been pruned (e.g., a pruned version of trace set "C2" is named "C2p"). Pruning reduces the number of clients by 5% (for trace C2) to 13% (for R2); the number of HTTP responses by 7% (for C2) to 23% (for R2, U3, and U4); and the peak request rate by 6% (for C2) to 11% (for R2).

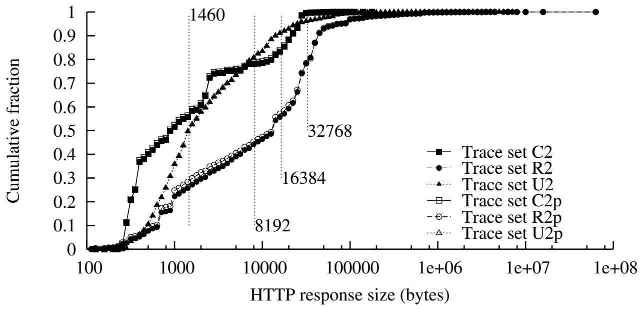


Figure 2: CDF of status-200 response sizes

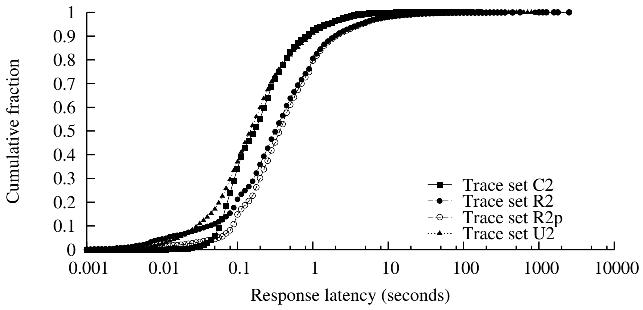


Figure 3: CDF of status-200 response latencies

The mean values in Table 1 do not convey the whole story. In Figures 2 and 3, respectively, we plot cumulative distributions for response size and latency for status-200 responses (These plots exclude the U3 and U4 traces, since these CDFs are nearly identical to those for the U2 trace; Figure 3 also excludes C2p and U2p, since these CDFs are nearly identical to those for the unpruned traces.)

The three traces in Figure 2 show quite different response size distributions. The responses in trace C2 seem somewhat smaller than has typically been reported for Web traces; the responses in trace R2 are a lot larger. (These differences also appear in the mean response sizes in Table 1.) Trace R2 is unusual, in part, because many users of the site download entire technical reports, which tend to be much larger than individual HTML or embedded-image files.

Figure 2 includes three vertical lines indicating the 8K byte, 16K byte, and 32K byte thresholds. Note that 8K is below the median size for R2, but above the median size for C2 and U2, but the median for all traces is well below 32K bytes.

			All HTTP status codes					status code = 200		
Trace name	Total Conns.	Total Clients	Total Resp. bytes	Total Resps.	mean resp. size (bytes)	mean req. rate	peak req. rate	Total Resp. bytes	Total Resps.	mean resp. size (bytes)
C2	323141	17627	3502M	1221961	3005	2.3/sec	193/sec	3376M	576887	6136
C2p (pruned)	281375	16671	3169M	1132030	2935	2.1/sec	181/sec	3053M	533582	5999
R2	33286	7730	1679M	50067	35154	0.1/sec	35/sec	1359M	40011	35616
R2p (pruned)	23296	6732	1319M	38454	35960	0.1/sec	31/sec	1042M	31413	34766
U2	261531	36170	5154M	909442	5942	1.7/sec	169/sec	4632M	580715	8363
U2p (pruned)	203055	33705	4191M	744181	5904	1.4/sec	152/sec	3754M	479892	8202
U3	278617	29843	5724M	987787	6076	11.4/sec	125/sec	5261M	637380	8655
U3p (pruned)	197820	26697	4288M	756994	5939	8.8/sec	117/sec	3940M	491497	8405
U4	326345	32047	6800M	1182049	6032	13.7/sec	139/sec	6255M	763545	8589
U4p (pruned)	230589	28628	5104M	902996	5926	10.5/sec	139/sec	4689M	588954	8347

Table 1: Overall trace characteristics

Figure 3 shows that response durations are significantly longer in the R2 trace than in the others, possibly because of the longer response sizes in R2.

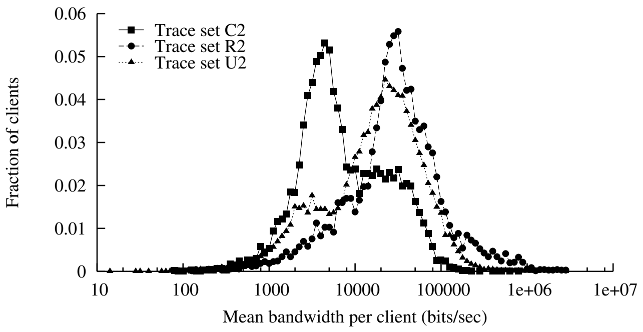


Figure 4: PDF of mean bandwidth per client

We calculated, for each distinct client, a mean bandwidth across all transfers for that client. Figure 4 shows the distributions; the pruned traces had similar distributions and are not shown. Trace C2 has a much larger fraction of low-bandwidth users than R2 or U2. The apparent slight excess of high-bandwidth clients in R2 might result from the larger responses in R2; larger transfers generally increase TCP's efficiency at using available bandwidth.

We also looked at the distribution of the TCP Maximum Segment Size (MSS) values in our traces. In trace R2, virtually all of the MSS values were at or close to the standard Ethernet limit (about 1460 bytes); in traces C2 and U2, about 95% of the MSS values were near the limit, with the rest mostly close to 512 bytes. Figure 2 shows a vertical line at 1460 bytes, indicating approximately where the dominant MSS value lies on the response-size distribution.

3.5 Trace anomalies

The monitoring architectures available to us differed at each of the collection sites. For example, at one of the sites port mirroring was used to copy packets from a monitored link to the mirrored link. At another site, separate links were tapped, one for packets bound for the Web server, the second for packets sent by the server. These monitoring infrastructures are subject to a variety of measurement errors:

- Port mirroring multiplexes bidirectional traffic from the monitored link onto the unidirectional mirror link. This can cause packets to appear in the trace in a different order than they arrived on the monitored link. Such reordering typically affects packets that occurred close together in time. For example, in the U2 trace, 10% of connections had the SYN and SYN|ACK packets in reverse order. Our Bro script corrects for this.
- Port mirroring temporarily buffers packets from the monitored link until they can be sent over the mirrored link. This buffer can overflow, causing packets to be dropped.
- Several of our environments have multiple network links that transfer packets to or from the Web server. Since we could not monitor all of these links, we did not capture all of the HTTP request/response transactions. In some cases we capture only half of the transaction (about 48% of the connections are affected by this in one trace).
- Ideally, a traced packet would be timestamped at the precise instant it arrives. However, trace-collection systems buffer packets at least briefly (often in several places) before attaching a timestamp, and packets are often collected at several nearby points (e.g., two packet monitors on both members of a pair of simplex links), which introduces timestamp errors due to imperfect clock synchronization. Erroneous timestamps could cause errors in our analysis by affecting either or both of our RTT estimates and our latency estimates.

We estimated the number of packets lost within our measurement system by watching for gaps in the TCP sequence numbers. This could overestimate losses (e.g., due to reordered packets) but the estimates, as reported in Table 2, are quite low.

Table 2 also shows our estimates (based on a separate Bro script) for packet retransmission rates on the path between client and server, implied by packets that cover part of the TCP sequence space we have already seen. Retransmissions normally reflect packet losses in the Internet, which would invalidate the model used in equation 1. Knowing these rates could help understand where the initial-RTT approach is applicable.

Note that Table 1 only includes connections with at least one complete HTTP response, while Table 2 includes all

Trace name	Total packets	Total Conns.	Measurement system lost pkts.	Retransmitted packets	Conns. w/ retransmitted packets	Conns. w/no pkts in one direction
C2	40474900	1182499	17017 (0.04%)	114911 (0.3%)	53906 (4.6%)	572052 (48.4%)
R2	2824548	43023	1238 (0.04%)	27140 (1.0%)	4478 (10.4%)	460 (1.1%)
U2	11335406	313462	5611 (0.05%)	104318 (0.9%)	26815 (8.6%)	17107 (5.5%)
U3	11924978	328038	2093 (0.02%)	89178 (0.7%)	26371 (8.0%)	14975 (4.6%)
U4	14393790	384558	5265 (0.04%)	110541 (0.8%)	30638 (8.0%)	18602 (4.8%)

Table 2: Packet loss rates

connections, including those that end in errors. We were only able to use 27% of the connections listed in Table 2 for C2, partly because we only saw packets in one direction for 48% of the connections. Our analysis script flagged another ≈20% of the C2 connections as **error_has_occurred**, possibly due to unknown problems in the monitoring infrastructure.

4 Predictions based on initial RTT: results

In this section, we summarize the results of our experiments on techniques to predict transfer latency using the initial RTT. We address these questions:

1. Does RTT *per se* correlate well with latency?
2. How well does equation 1 predict latency?
3. Can we improve on equation 1?
4. What is the effect of modem compression?
5. How sensitive are the predictions to parameter choices?

There is no single way to define what it means for a latency predictor to provide "good" predictions. We evaluate prediction methods using several criteria, including the correlation between predicted and measured latencies, and the mean and median of the difference between the actual and predicted latencies.

4.1 Does RTT itself correlate with latency?

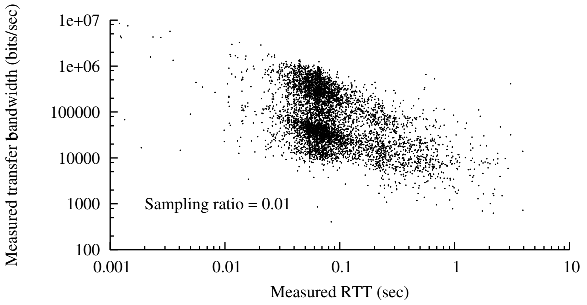


Figure 5: Scatter plot of bandwidth vs. RTT, trace C2

Perhaps it is unnecessary to invoke the full complexity of equation 1 to predict latency from RTT. To investigate this, we examined the correlation between RTT *per se* and either bandwidth or latency.

For example, Figure 5 shows a scatter plot of bandwidth vs. initial RTT, for all status-200 responses in trace C2. (In order to avoid oversaturating our scatter plots, we randomly sampled the actual data in each plot; the sampling

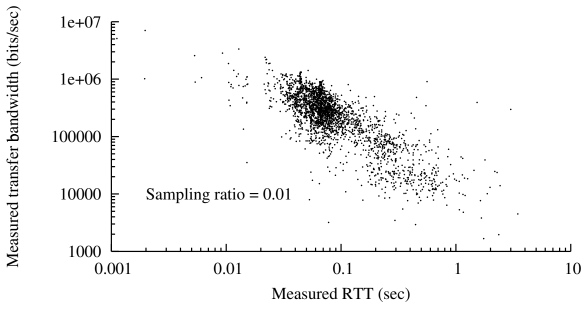


Figure 6: BW vs. RTT, trace C2, 1 MSS < length < 32KB

ratios are shown in the figures.) The graph shows an apparent weak correlation between initial RTT and transfer bandwidth. Corresponding scatter plots for R2, U2, U3, and U4 show even weaker correlations.

We found a stronger correlation if we focused on transfer lengths above one MSS and below 32K bytes, as shown in Figure 6. Our technique for measuring latency is probably least accurate for responses below one MSS (i.e., those sent in just one packet). Also, single-packet responses may suffer excess apparent delay (as measured by when the server receives the final ACK) because of delayed acknowledgment at the client. In our subsequent analyses, we exclude responses with lengths of one MSS or less because of these measurement difficulties. The 32KB threshold represents one plausible choice for defining a "short" transfer.

Trace name	Samples included	Correlation w/bandwidth	Correlation w/latency
C2	140234 (24.3%)	-0.352	0.511
C2p	129661 (24.3%)	-0.370	0.508
R2	7500 (18.7%)	-0.112	0.364
R2p	5519 (17.6%)	-0.054	0.418
U2	218280 (37.6%)	-0.163	0.448
U2p	181180 (37.8%)	-0.178	0.458
U3	234591 (36.8%)	-0.181	0.421
U3p	181276 (36.9%)	-0.228	0.427
U4	283993 (37.2%)	-0.179	0.364
U4p	219472 (37.3%)	-0.233	0.411

(a) 1 MSS < length < 8KB

Trace name	Samples included	Correlation w/bandwidth	Correlation w/latency
C2	261931 (45.4%)	-0.325	0.426
C2p	238948 (44.8%)	-0.339	0.426
R2	20546 (51.4%)	-0.154	0.348
R2p	15407 (49.0%)	-0.080	0.340
U2	312090 (53.7%)	-0.165	0.392
U2p	258049 (53.8%)	-0.179	0.401
U3	336443 (52.8%)	-0.162	0.263
U3p	259028 (52.7%)	-0.215	0.276
U4	414209 (54.2%)	-0.167	0.287
U4p	320613 (54.4%)	-0.215	0.343

(b) 1 MSS < length < 32KB

Table 3: Correlations: RTT vs. either bandwidth or latency

For a more quantified evaluation of this simplistic approach, we did a statistical analysis using a simple R [18] program. The results are shown in Table 3(a) and (b), for lengths limited to 8K and 32K bytes, respectively.

The tables show rows for both pruned and unpruned versions of the five basic traces. We included only status-200 responses whose length was at least one MSS; the "samples included" column shows that count for each trace. The last two columns show the computed correlation between initial RTT and either transfer bandwidth or transfer latency. (The bandwidth correlations are negative, because this is an inverse relationship.)

For the data set including response lengths up to 32K bytes, none of these correlations exceeds 0.426, and many are much lower. If we limit the response lengths to 8K bytes, the correlations improve, but this also eliminates most of the samples.

We tried excluding samples with an initial RTT value above some quantile, on the theory that high RTTs correlate with lossy network paths; this slightly improves RTT vs. bandwidth correlations (for example, excluding records with an RTT above 281 msec reduces the number of 32K-or-shorter samples for R2 by 10%, and improves that correlation from -0.154 to -0.302) but it actually worsens the latency correlations (for the same example, from 0.348 to 0.214).

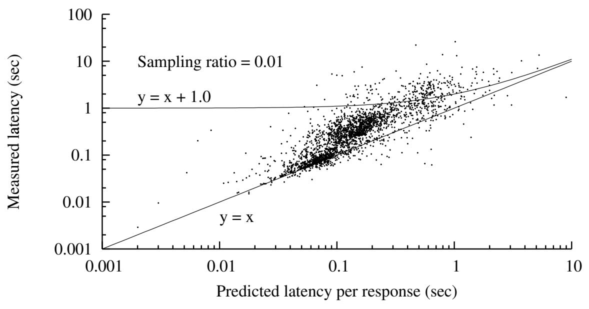
Note that, contrary to our expectation that traces pruned of proxies and robots would be less predictable, in Table 3 this seems true only for the R2 trace; in general, pruning seems to slightly improve predictability. In fact, while we present results for both pruned and unpruned traces throughout the paper, we see no consistent difference in predictability.

4.2 Does equation 1 predict latency?

Although we did not expect RTT to correlate well with latency, we might expect better results from the sophisticated model derived by Cardwell *et al.* [2]. They validated their model (equation 1 is a simplified version) using HTTP transfers over the Internet, but apparently used only "well-connected" clients and so did not probe its utility for poorly-connected clients. They also used RTT estimates that included more samples than just each connection's initial RTT.

We therefore analyzed the ability of equation 1 to predict transfer bandwidths and latencies using only the initial RTT, and with the belief that our traces include some poorly-connected clients.

Figure 7 shows an example scatter plot of measured latency vs. predicted latency, for trace C2. Again, we include only status-200 responses at least one MSS in length. We have superimposed two curves on the plot. (Since this is a log-log plot, most linear equations result in curved lines.) Any point above the line $y = x$ represents an under-



$\gamma = 2.0, w_1 = 4$

Figure 7: Real vs. predicted latency, trace C2

prediction of latency; underpredictions are generally worse than overpredictions, if (for example) we want to avoid exposing Web users to unexpectedly long downloads. Most of the points in the plot are above that line, but most are below the curve $y = x + 1.0sec$, implying that most of the overpredictions (in this example) are less than 1 sec in excess. However, a significant number are many seconds too high.

We extended our R program to compute statistics for the predictive ability of equation 1. For each status-200 trace record with a length between one MSS and 32K bytes, we used the equation to predict a latency, and then compared this to the latency recorded in the trace record. We then computed the correlation between the actual and predicted latencies. We also computed a residual error value, as the difference between the actual and predicted latencies. Table 4 shows the results from this analysis, using $\gamma = 1.5$ and $w_1 = 1$, a parameter assignment that worked fairly well across all five traces.

Trace name	Samples included	Correlation w/latency	Median residual	Mean residual
C2	261931 (45.4%)	0.581	-0.017	0.164
C2p	238948 (44.8%)	0.584	-0.015	0.176
R2	20546 (51.4%)	0.416	-0.058	0.261
R2p	15407 (49.0%)	0.421	-0.078	0.272
U2	312090 (53.7%)	0.502	-0.022	0.110
U2p	258049 (53.8%)	0.519	-0.024	0.124
U3	336443 (52.8%)	0.334	-0.018	0.152
U3p	259028 (52.7%)	0.353	-0.016	0.156
U4	414209 (54.2%)	0.354	-0.013	0.141
U4p	320613 (54.4%)	0.425	-0.010	0.136

Residual values are measured in seconds; 1 MSS $<$ length $<$ 32KB

Table 4: Quality of predictions based on equation 1

In Table 4, the median residuals are always negative, implying that equation 1 overestimates the transfer latency more often than it underestimates it. However, the mean residuals are always positive, because the equation's underestimates are more wrong (in absolute terms) than its overestimates. The samples in Figure 7 generally follow a line with a steeper slope than $y = x$, suggesting that equation 1 especially underestimates higher latencies.

One possible reason is that, for lower-bandwidth links, RTT depends on packet size. For a typical 56Kb/s modem link, a SYN packet will see an RTT somewhat above

100 msec, while a 1500 byte data packet will see an RTT several times larger. This effect could cause equation 1 to underestimate transfer latencies.

4.3 Can we improve on equation 1?

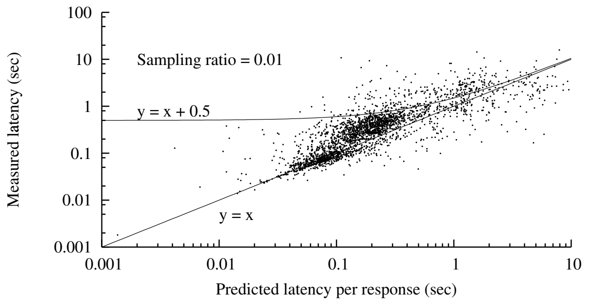
Given that equation 1 seems to systematically underestimate higher latencies, exactly the error that we want to avoid, we realized that we could modify the equation to reduce these errors.

We experimented with several modifications, including a linear multiplier, but one simple approach is:

```
function ModifiedEqnOne(RTT, MSS, Length, $w_1$,
                        $\gamma$, CompWeight)
temp = EquationOne(RTT, MSS, Length, $w_1$, $\gamma$);
return(temp + (temp*temp*CompWeight));
```

That is, we "overpredict" by a term proportional to the square of the original prediction. This is a *heuristic*, not the result of rigorous theory.

We found by trial and error that a proportionality constant, or "compensation weight," $CompWeight = 2.25$ worked best for C2, but $CompWeight = 1.75$ worked better for R2 U2, and $CompWeight = 1.25$ worked best for U3 and U4. For all traces, $\gamma = 2$ got the best results, and we set $w_1 = 4$ for C2 and U2, and $w_1 = 3$ for R2, U3, and U4. We discuss the sensitivity to these parameters in Section 4.5.



$\gamma = 2.0$, $w_1 = 4$, $CompWeight = 2.25$

Figure 8: Modified prediction results, trace C2

Figure 8 shows how the modified prediction algorithm systematically overpredicts at higher latencies, while not significantly changing the accuracy for lower latencies. (For example, in this figure, $CompWeight = 2.25$; if equation 1 predicts a latency of 0.100 seconds, the modified prediction will be 0.1225 seconds). However, even the modified algorithm significantly underpredicts a few samples; we do not believe we can avoid this, especially for connections that suffer packet loss (see Table 2).

Table 5 shows that the modifications to equation 1 generally worsen the correlations, compared to those in Table 4, but definitely improves the residuals – the median error is always less than 100 msec, and the mean error is less than 15 msec, except for traces U3p and U4p (our parameter choices were tuned for the unpruned traces).

Trace name	Samples included	Correlation w/latency	Median residual	Mean residual
C2	261931 (45.4%)	0.417	0.086	-0.002
C2p	238948 (44.8%)	0.423	0.092	-0.006
R2	20546 (51.4%)	0.278	0.015	0.002
R2p	15407 (49.0%)	0.311	0.019	0.013
U2	312090 (53.7%)	0.386	0.053	0.010
U2p	258049 (53.8%)	0.402	0.056	0.001
U3	336443 (52.8%)	0.271	0.034	0.011
U3p	259028 (52.7%)	0.302	0.036	-0.020
U4	414209 (54.2%)	0.279	0.035	0.003
U4p	320613 (54.4%)	0.337	0.038	-0.033

Residual values are measured in seconds; 1 MSS $<$ length $<$ 32KB

Table 5: Predictions based on modified equation 1

4.4 Text content and modem compression

Many people still use dialup modems. It has been observed that to accurately model path bandwidth, one must account for the compression typically done by modems [3]. However, most image Content-Types are already compressed, so this correction should only be done for text content-types.

HTTP responses normally carry a MIME Content-Type label, which allowed us to analyze trace subsets for "text/*" and "image/*" subsets. Table 6 shows the distribution of these coarse Content-Type distinctions for the traces.

We speculated that the latency-prediction model of equation 1, which incorporates the response length, could be further improved by reducing this length value when compression might be expected. (A server making predictions knows the Content-Types of the responses it plans to send. Some servers might use a compressed content-coding for text responses, which would obviate the need to correct predictions for those responses for modem compression. We found no such responses in our traces.)

We cannot directly predict either the compression ratio (which varies among responses and among modems) nor can we reliably determine which clients in our traces used modems. Therefore, for feasibility of analysis our model assumes a constant compressibility factor for text responses, and we tested several plausible values for this factor. Also, we assumed that an RTT below 100 msec implied a non-modem connection, and RTTs above 100 msec implied the *possible* use of a modem. In a real system, information derived from the client address might identify modem-users more reliably. (In Section 5 we classify clients using hostnames; but this might add too much DNS-lookup delay to be effective for latency prediction.)

Table 7 shows results for text content-types only, using the modified prediction algorithm based on equation 1, but without correcting for possible modem compression. We set $\gamma = 2.0$ for C2 and U2, and $\gamma = 1.5$ for R2, U3, and U4; $w_1 = 2$ for C2 and $w_1 = 3$ for the other traces; and $CompWeight = 1$ for all traces. (We have not tested a wide range of $CompWeight$ values to see if text content-types would benefit from a different $CompWeight$.) Compared to the results for all content types (see Table 5), the residuals for text-only samples are generally higher.

Content-type	C2	R2	U2	U3	U4
Unknown	3 (0.00%)	26 (0.06%)	178 (0.03%)	157 (0.02%)	144 (0.02%)
TEXT/*	122426 (21.22%)	23139 (57.83%)	85180 (14.67%)	92108 (14.45%)	107958 (14.14%)
IMAGE/*	454458 (78.78%)	13424 (33.55%)	465160 (80.10%)	507330 (79.60%)	607520 (79.57%)
APPLICATION/*	0 (0.00%)	3410 (8.52%)	29733 (5.12%)	37581 (5.90%)	47765 (6.26%)
VIDEO/*	0 (0.00%)	4 (0.01%)	17 (0.00%)	10 (0.00%)	5 (0.00%)
AUDIO/*	0 (0.00%)	8 (0.02%)	446 (0.08%)	194 (0.03%)	140 (0.02%)

Table 6: Counts and frequency of content-types (excluding some rarely-seen types)

Trace name	Samples included	Correlation w/latency	Median residual	Mean residual
C2	118217 (96.6%)	0.442	0.142	0.002
C2p	106120 (96.4%)	0.449	0.152	-0.003
R2	12558 (54.3%)	0.288	0.010	0.066
R2p	8760 (50.2%)	0.353	0.017	0.105
U2	70924 (83.3%)	0.292	0.100	0.073
U2p	56661 (83.0%)	0.302	0.110	0.066
U3	76714 (83.3%)	0.207	0.063	-0.021
U3p	56070 (83.2%)	0.198	0.072	-0.099
U4	90416 (83.8%)	0.281	0.065	-0.034
U4p	65708 (83.8%)	0.359	0.078	-0.122

Residual values are measured in seconds; 1 MSS $<$ length $<$ 32KB

Table 7: Predictions for text content-types only

Trace name	Samples included	Compression factor	Correlation w/latency	Median residual	Mean residual
C2	118217	1.0	0.442	0.142	0.002
C2p	106120	1.0	0.449	0.152	-0.003
R2	12558	4.0	0.281	0.013	0.002
R2p	8760	4.0	0.345	0.021	0.044
U2	70924	3.0	0.295	0.083	0.008
U2p	56661	3.0	0.306	0.096	-0.004
U3	76714	4.0	0.208	-0.002	0.001
U3p	56070	4.0	0.201	0.003	-0.063
U4	90416	4.0	0.277	-0.000	-0.011
U4p	65708	4.0	0.353	0.007	-0.083

Residual values are measured in seconds; 1 MSS $<$ length $<$ 32KB

Table 8: Predictions for text with compression

Table 8 shows results for text content-types when we assumed that modems compress these by the factor shown in the third column. Note that for C2 and C2p, we got the best results using a compression factor of 1.0 – that is, without correcting for compression. For the other traces, correcting for compression did give better results. Here we set the other parameters as: $\gamma = 2$ (except for U3 and U4, where $\gamma = 1.5$ worked best), $w_1 = 1$ (except for C2, where $w_1 = 2$ worked best), and $CompWeight = 1.0$ (except for R2, where $CompWeight = 2.25$ worked best). We experimented with assuming that the path did not involve a modem (and thus should not be corrected for compression) if the initial RTT was under 100 msec, but for R2 and U2 it turned out that we got the best results when we assumed that all text responses should be corrected for compression.

Table 8 shows that, except for trace C2, correcting for modem compression improves the mean residuals over those in Table 7. We have not evaluated the use of compression factors other than integers between 1 and 4, and we did not evaluate a full range of $CompWeight$ values for this section.

Image content As shown in Table 6, image content-types dominate most of the traces, except for R2. Also, Web site designers are more likely to have choices between rich and simple content for image types than for text types. (Designers often include optional "Flash" animations, but we found almost no Flash content in C2 and R2, and relatively little in U2, U3, and U4.) We therefore compared the predictability of transfer latencies for image content-types, but found no clear difference compared to the results for all content in general.

4.5 Sensitivity to parameters

How sensitive is prediction performance to the parameters γ, w_1, and $CompWeight$? That question can be framed in several ways: how do the results for one server vary with parameter values? If parameters are chosen based on traces from server X, do they work well for server Y? Are the optimal values constant over time, client sub-population, content-type, or response length? Do optimal parameter values depend on the performance metric? For reasons of space, we focus on the first two of these questions.

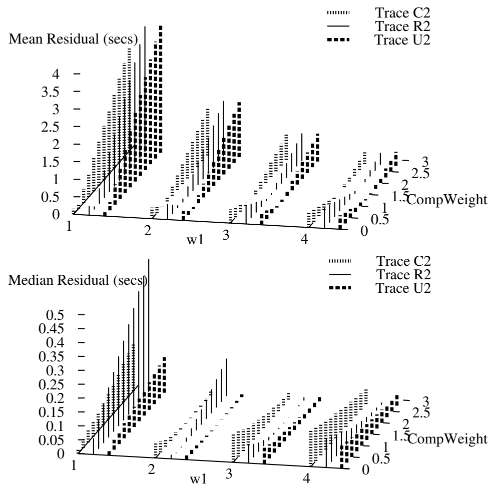
Figure 9 shows how the absolute values of the mean and median residuals vary with γ, w_1, and $CompWeight$ for traces C2, R2, and U2. The optimal parameter choice depends on whether one wants to minimize the mean or the median; for example, for R2, $\gamma = 2.0$, $w_1 = 3$, and $CompWeight = 1.75$ yields an optimal mean of 1.5 msec (and a median of 15 msec). The median can be further reduced to 0.2 msec, but at the cost of increasing the mean to over half a second.

Figure 9 also shows how the optimal parameters vary across several traces. (Results for traces U3 and U4 are similar to those for U2, and are omitted to reduce clutter.) It appears that no single choice is optimal across all traces, although some choices yield relatively small mean and medians for many traces. For example, $\gamma = 2$, $w_1 = 3$, and $CompWeight = 1.25$ yields optimal or near-optimal mean residuals for U2, U3, and U4, and decent results for C2.

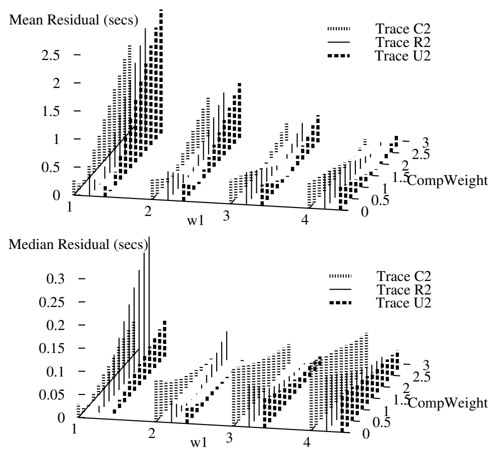
4.6 Training and testing on different data

The results we have presented so far used parameter choices "trained" on the same data sets as our results were tested on. Since any real prediction system requires advance training, we also evaluated predictions with training and testing on different data sets.

Our trace collection was not carefully designed in this regard; we have no pairs of data sets that are completely



Mean and median for $\gamma = 1.5$



Mean and median for $\gamma = 2.0$

Figure 9: Sensitivity of residual absolute values to parameters, 1 MSS $<$ length $<$ 32KB (note different y-axis scales)

identical and adjacent in time. For the C2, R2, and U2 data sets, we chose the first three days as the training data set, and the last four days as the testing data set. However, because we collected data at different hours on each day, and because there are day-of-week differences between the training and testing sets (the testing sets includes two weekend days), we suspect that these pairs of data sets might not be sufficiently similar. We also used the U3 data set to train parameters that we then tested on the U4 data set; these two traces are more similar to each other.

	Trained parameters				Testing results		
Trace name	γ	w_1	Comp Weight	residual in training	rank (of 96)	residual	best resid.
C2	2.0	4	2.50	-0.000	15	-0.098	-0.004
C2p	2.0	3	1.75	-0.004	12	-0.089	-0.002
R2	1.5	4	1.50	-0.004	20	0.136	0.000
R2p	1.5	3	1.00	0.003	16	0.125	0.003
U2	1.5	4	1.50	0.001	10	-0.072	0.012
U2p	2.0	2	0.75	-0.004	9	-0.081	-0.002
U3U4	2.0	2	0.75	-0.007	3	-0.013	0.003
U3U4p	2.0	1	0.25	0.000	2	-0.013	-0.010

Residual values are measured in seconds; 1 MSS $<$ length $<$ 32KB

Table 9: Training and testing on different data

Table 9 shows results for training vs. testing. We tested and trained with 96 parameter combinations, based on the two possible choices for γ, the four choices for w_1, and twelve equally-spaced choices for $CompWeight$. The **trained parameters** are those that minimize the absolute value of the mean **residual in training**. The columns under **testing results** show how the results using the trained parameters rank among all of the testing results, the mean residual when using those parameters, and the residual for the best possible parameter combination for the testing data.

These results suggest that the degree to which training can successfully select parameter values might vary significantly from site to site. Based on our traces, we would have had the most success making useful predictions at the University site (U3-U4), and the least success at the Research site (R2).

However, the difference in "trainability" that we observed might instead be the result of the much closer match between the U3 and U4 datasets, compared to the time-of-day and day-of-week discrepancies in the other train/test comparisons. For C2, R2, and U2, we tried training just on one day (Tue., May 4, 2004) and testing on the next day, and got significantly better trainability (except for R2p, which was slightly worse) than in Table 9; this supports the need to match training and testing data sets more carefully.

4.7 A server's decision algorithm

To understand how a server might use the initial-RTT approach in practice, Figure 10 presents pseudo-code for generating predictions. (This example is in the context of a Web server adapting its content based on predicted transfer latency, but the basic idea should apply to other contexts.) If the server has $N > 1$ choices of response length for a given request, it would invoke *PredictLatency* $N - 1$ times, starting with the largest candidate and moving down in size, until it either finds one with a small-enough predicted latency, or has only one choice left. The first three arguments to the *PredictLatency* function (RTT, MSS, and client IP address) are known as soon as the connection is open. The last two (response content type and length) are specific to a candidate response that the server might send.

The function *ProbablyDialup*, not shown here, is a heuristic to guess whether a client is connected via a modem (which would probably compress text responses). It could simply assume that RTTs above 100 msec are from dialups, or it could use additional information based on the client's DNS name or AS (Autonomous System) number to identify likely dialups.

```
1.   function
         PredictLatency(RTT, MSS, ClientIP, ContentType, Length)

2.   if (ProbablyDialup(ClientIP, RTT)
             and (ContentType == TEXT)) then
3.       effectiveLength := Length/TextCompressionFactor;
4.   else
5.       effectiveLength := Length;
6.   end

7.   if (length > maxPredictableLength) then
8.       return(NO_PREDICTION); /* probably leaves slow-start */
9.   else if (length < MSS) then
10.      return(NO_PREDICTION); /* only one data packet to send */
11.  end

12.  return(ModifiedEqnOne(RTT, MSS, Length, $w_1$, $\gamma$,
             CompWeight));
```

$TextCompressionFactor$ is an estimate of the mean compression ratio for modems on text files;
$CompWeight$. w_1, and γ could themselves vary based on the server's observation of recent history, the ContentType, etc.

Figure 10: Pseudo-code for the decision algorithm

5 Detecting dialups

We speculated that a server could discriminate between dialups and non-dialups using clues from the client's "fully-qualified domain name" (FQDN). We obtained FQDNs for about 75% of the clients in the U4 trace, and then grouped them according to clues in the FQDNs that implied geography and network technology. Note that many could not be categorized by this method, and some categorizations are certainly wrong.

Category	Conns.	5%ile	median	mean	95%ile
By geography					
All	326359	0.008	0.069	**0.172**	**0.680**
N. America	35972	0.003	0.068	**0.124**	**0.436**
S. America	2372	**0.153**	**0.229**	**0.339**	**0.882**
Europe	12019	**0.131**	**0.169**	**0.262**	**0.717**
Asia-Pacific	9176	**0.165**	**0.267**	**0.373**	**0.885**
Africa	2027	**0.206**	**0.370**	**0.486**	**1.312**
"Dialup" in FQDN					
All	11478	**0.144**	**0.350**	**0.664**	**2.275**
Regional	5977	**0.133**	**0.336**	**0.697**	**2.477**
Canada	1205	**0.208**	**0.460**	**0.751**	**2.060**
US	575	**0.189**	**0.366**	**0.700**	**2.210**
Europe	566	**0.183**	**0.216**	**0.313**	**0.861**
"DSL" in FQDN					
All	59211	0.003	0.023	0.060	**0.210**
Local	1816	0.011	0.022	0.034	0.085
Regional	47600	0.009	0.018	0.032	0.079
US	1053	0.071	0.085	**0.117**	**0.249**
Europe	118	**0.148**	**0.162**	**0.178**	**0.313**
"Cable" in FQDN					
All	6599	0.039	0.077	**0.132**	**0.338**
Canada	2741	0.039	0.055	0.088	**0.222**
US	585	0.072	0.086	0.094	**0.127**
Europe	600	**0.143**	**0.155**	**0.176**	**0.244**

Times in seconds; **bold** entries are > 0.1 sec.

Table 10: RTTs by geography and connection type

Table 10 shows how initial RTTs vary by geography and connection type. For the connections that we could categorize, at least 95% of "dialup" connections have RTTs above 100 msec, and most "cable" and "DSL" connections have RTTs below 200 msec. These results seem unaffected by further geographical subdivision, and support the hypothesis that a threshold RTT between 100 and 200 msec would discriminate fairly well between dialup and non-dialup connections. We do not know if these results apply to other traces.

6 Predictions from previous bandwidths: results

In this section, we compare how well prediction based on variants of equation 1 compares with predictions from the older recent-transfers approach. We address these questions:

1. How well can we predict latency from previous bandwidth measurements?
2. Does a combination of the two approaches improve on either individual predictor?

Note that the recent-transfers approach cannot specifically predict the latency for the very first transfer to a given client, because the server has no history for that client. This is a problem if the goal is to provide the best user experience for a client's initial contact with a Web site. For initial contacts, a server using the recent-transfers approach to predict latency has several options, including:

- Make no prediction.
- "Predict" the latency based on history across all previous clients; for example, use an exponentially smoothed mean of all previous transfer bandwidths.
- Assume that clients with similar network locations, based on routing information, have similar bandwidths; if a new client belongs to "cluster" of clients with known bandwidths, use history from that cluster to make a prediction. Krishnamurthy and Wang [8] describe a technique to discover clusters of client IP addresses. Krishnamurthy and Wills [9] then showed, using a set of chosen Web pages with various characteristics, that clustering pays off in prediction accuracy improvements ranging up to about 50%. We speculate that this approach would also work for our traces.
- Use the initial-RTT technique to predict a client's first-contact latency, and use the recent-transfers technique to predict subsequent latencies for each client. We call this the *hybrid* technique.

We first analyze the purest form of recent-transfers (making no prediction for first-contact clients), and then consider the mean-of-all-clients and hybrid techniques.

6.1 Does previous bandwidth predict latency?

We did a statistical analysis of the prediction ability of several variants of the pure recent-tranfers technique. In each case, we made predictions and maintained history only for transfer lengths of at least one MSS. Table 11

		Correlation with		
		most	mean	weighted
Trace	Samples	recent	previous	mean
name	included	bandwidth	bandwidth	bandwidth
C2	262165 (45.4%)	0.674	0.742	**0.752**
C2p	238957 (44.8%)	0.658	0.732	**0.737**
R2	24163 (60.4%)	0.589	0.655	**0.666**
R2p	17741 (56.5%)	0.522	0.543	**0.579**
U2	310496 (53.5%)	0.527	0.651	**0.654**
U2p	254024 (52.9%)	0.437	**0.593**	0.561
U3	341968 (53.7%)	0.495	0.627	**0.638**
U3p	260470 (53.0%)	0.508	**0.659**	0.625
U4	421867 (55.3%)	0.521	**0.690**	0.647
U4p	323811 (55.0%)	0.551	**0.690**	0.656

Best correlation for each trace shown in **bold**

Table 11: Correlations: measured vs. recent bandwidths

shows the results. The first two columns show the trace name and the number of samples actually used in the analysis. The next three columns show the correlations between the bandwidth (not latency) in a trace record and, respectively, the most recent bandwidth for the same client, the mean of previous bandwidths for the client, and the exponential weighted mean $X_i = \alpha \cdot X_{i-1} + (1 - \alpha) measurement_i$. We followed Krishnamurthy *et al.* [10] in using $\alpha = 0.7$, although other values might work better for specific traces.

These results suggest that some form of mean is the best variant for this prediction technique; although the choice between simple means and weighted means varies between traces, these always outperform predictions based on just the most previous transfer. Since Krishnamurthy *et al.* [10] preferred the weighted mean, we follow their lead for the rest of this paper.

Pruning the traces, as we had expected, does seem to decrease the predictability of bandwidth values, except for the U3 and U4 traces. This effect might be magnified for the recent-transfers technique, since (unlike the initial-RTT technique) it relies especially on intra-client predictability.

Table 11 showed correlations between bandwidth measurements and predictions. To predict a response's latency, one can combine a bandwidth prediction with the known response length. Table 12 shows how well the weighted mean bandwidth technique predicts latencies. Table 12(a) includes responses with length at least one MSS; Table 12(b) excludes responses longer than 32 Kbytes. Because short responses and long responses may be limited by different parameters (RTT and bottleneck bandwidth, respectively), we hypothesized that it might not make sense to predict short-response latencies based on long-response history. Indeed, the residuals in Table 12(b) are always better than the corresponding values in Table 12(a), although the correlations are not always improved.

The correlations in Table 12(a) are better than those from the modified equation 1 as shown in Table 5, except for trace U4. However, the mean residuals in Table 12 are much larger in magnitude than in Table 5; it might be possible to correct the bandwidth-based predictor to fix this.

Trace name	Samples included	Correlation w/latency	Median residual	Mean residual
C2	262165 (45.4%)	0.514	-0.042	-0.502
C2p	238957 (44.8%)	0.515	-0.046	-0.529
R2	24163 (60.4%)	0.525	-0.066	-4.100
R2p	17741 (56.5%)	0.560	-0.140	-5.213
U2	310496 (53.5%)	0.475	-0.028	-1.037
U2p	254024 (52.9%)	0.460	-0.033	-1.142
U3	341968 (53.7%)	0.330	-0.025	-1.138
U3p	260470 (53.0%)	0.374	-0.029	-1.288
U4	421867 (55.3%)	0.222	-0.021	-0.957
U4p	323811 (55.0%)	0.251	-0.024	-1.111

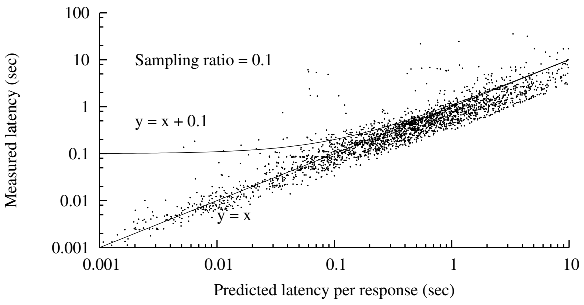
(a) 1 MSS $<$ length

Trace name	Samples included	Correlation w/latency	Median residual	Mean residual
C2	256943 (44.5%)	0.516	-0.038	-0.485
C2p	234160 (43.9%)	0.516	-0.043	-0.512
R2	17445 (43.6%)	0.317	-0.018	-0.779
R2p	12741 (40.6%)	0.272	-0.054	-0.959
U2	287709 (49.5%)	0.256	-0.020	-0.407
U2p	235481 (49.1%)	0.247	-0.024	-0.454
U3	314965 (49.4%)	0.447	-0.017	-0.300
U3p	239843 (48.8%)	0.484	-0.020	-0.336
U4	390981 (51.2%)	0.338	-0.015	-0.274
U4p	299905 (50.9%)	0.312	-0.017	-0.314

(a) 1 MSS $<$ length $<$ 32KB

Table 12: Latency prediction via weighted mean bandwidth

The previous-bandwidth approach consistently overpredicts latency, which in some applications might be better than underprediction. Figure 11 shows an example scatter plot, for R2. In the Web-server content adaptation application, excessive overprediction increases the chances that a well-connected user will fail to receive rich content, although this is less harmful than sending excessive content to a poorly-connected user.



1 MSS $<$ length

Figure 11: Real vs. bandwidth-predicted latency, trace R2

6.2 Combining predictors

Given that the initial-RTT approach seems more accurate at predicting first-contact latencies, for many thresholds, than the recent-transfers approach, we speculated that a hybrid of the two predictors might yield the best results. This hybrid would use the modified equation 1 predictor for a client's first-contact transfer, and the smoothed mean of the client's previous bandwidths for its subsequent transfers.

We found that the overall (all-transfers) accuracy of this hybrid is nearly indistinguishable from the overall accuracy of the recent-transfers approach because, as the statistics in Table 1 imply, only a small fraction of transfers in our traces are first contacts.

7 Summary and conclusions

We conducted a study, based on traces from several different user communities, to demonstrate how well two different approaches can predict the latency of short TCP transfers. We found that by making a minor modification to a previously-described formula, we could greatly reduce its absolute prediction errors. We showed that predictions based on observation of past history generally yield better overall correlations than our formula-based predictor, but the formula-based predictor has lower mean prediction errors. We also show that the formula-based predictor could be improved to handle the specific case of text content, where modem-based compression can affect latency. Finally, we reported results from a study on the relationship between round-trip time and the use of modems, suggesting that this relationship might be exploited to improve prediction accuracy.

This paper has not quantified how much a real application, such as a Web server, could improve end-to-end performance by using our prediction techniques. Our technical report [1] provides some additional analysis of this and other details that do not fit here.

Acknowledgments

We would like to thank Vern Paxson, for his help with Bro and especially for writing the first draft of our Bro script and for improving Bro to meet our needs; Jerry Shan, for lots of help with statistical analysis; and Mike Rodriquez, Oliver Spatscheck, and others for their help in support of data collection. We thank Rick Jones, Zhuoqing Morley Mao, Dejan Milojicic, Mehul Shah, Chris Tuttle, Carey Williamson, and the anonymous reviewers for their helpful comments.

References

[1] ARLITT, M., KRISHNAMURTHY, B., AND MOGUL, J. C. Predicting short-transfer latency from TCP arcana: extended version. Tech. Report HPL-2005-137, HP Labs, Sep 2005.

[2] CARDWELL, N., SAVAGE, S., AND ANDERSON, T. Modeling TCP latency. In *Proc INFOCOM (3)* (Tel Aviv, Mar. 2000), pp. 1742–1751.

[3] CHENG, Y.-C., HÖLZLE, U., CARDWELL, N., SAVAGE, S., AND VOELKER, G. M. Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying. In *Proc. USENIX Annual Tech. Conf.* (Boston, MA, June 2004), pp. 87–98.

[4] CHESHIRE, S. Latency and the quest for interactivity, Nov. 1996. http://www.stuartcheshire.org/papers/LatencyQuest.ps.

[5] CHESHIRE, S. Latency survey results (for "It's the Latency, Stupid"). http://www.stuartcheshire.org/rants/LatencyResults.html, 1996.

[6] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. RFC 2616: Hypertext transfer protocol—HTTP/1.1, June 1999.

[7] HE, Q., DOVROLIS, C., AND AMMAR, M. On the Predictability of Large Transfer TCP Throughput. In *Proc. SIGCOMM* (Philadelphia, PA, Aug 2005).

[8] KRISHNAMURTHY, B., AND WANG, J. On network-aware clustering of Web clients. In *SIGCOMM* (Stockholm, Aug. 2000), pp. 97–110.

[9] KRISHNAMURTHY, B., AND WILLS, C. E. Improving Web performance by client characterization driven server adaptation. In *Proc. WWW2002* (Honolulu, HI, May 2002), pp. 305–316.

[10] KRISHNAMURTHY, B., WILLS, C. E., ZHANG, Y., AND VISHWANATH, K. Design, implementation, and evaluation of a client characterization driven web server. In *Proc. WWW2003* (Budapest, May 2003), pp. 138–147.

[11] LAI, K., AND BAKER, M. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proc. USITS* (San Francisco, CA, Mar 2001), pp. 123–134.

[12] MICROSOFT CORP. Knowledge Base Article 254337: Internet Explorer Sends Two GET Requests for the Same Data When the MIME Handler Is an ActiveX Control, 2000. http://support.microsoft.com/default.aspx?scid=kb;en-us;254337.

[13] MICROSOFT CORP. Exchange 2003 Design and Architecture at Microsoft. http://www.microsoft.com/technet/itsolutions/msit/deploy/ex03atwp.mspx, Aug 2003.

[14] MICROSOFT CORP. Knowledge Base Article 293792: Three GET Requests Are Sent When You Retrieve Plug-in Served Content, 2003. http://support.microsoft.com/default.aspx?scid=kb;en-us;293792.

[15] MOGUL, J., AND BRAKMO, L. Method for dynamically adjusting multimedia content of a web page by a server in accordance to network path characteristics between client and server. U.S. Patent 6,243,761, June 2001.

[16] MOGUL, J., BRAKMO, L., LOWELL, D. E., SUBHRAVETI, D., AND MOORE, J. Unveiling the transport API. In *Proc. 2nd Workshop on Hot Topics in Networks* (Cambridge, MA, November 2003).

[17] PAXSON, V. Bro: A system for detecting network intruders in real-time. *Computer Networks 31*, 23-24 (Dec. 1999), 2435–2463.

[18] R DEVELOPMENT CORE TEAM. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3.

[19] SCHROEDER, B., AND HARCHOL-BALTER, M. Web servers under overload: How scheduling can help. *ACM Trans. on Internet Techologies 6*, 1 (Feb. 2006).

[20] SU, J., CHIN, A., POPIVANOVA, A., GOEL, A., AND DE LARA, E. User mobility for opportunistic ad-hoc networking. In *Proc. WMCSA* (Lake District, UK, Dec 2004), pp. 41–50.

Novel approaches to end-to-end packet reordering measurement

Xiapu Luo and Rocky K. C. Chang
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, SAR, China
Email: $\{$*csxluo*$|$*csrchang*$\}$*@comp.polyu.edu.hk*

Abstract

By providing the best-effort service, the Internet Protocol (IP) does not maintain the same order of packets sent out by a host. Therefore, due to the route change, parallelism inside a switch, and load-balancing schemes, IP packets can be received in an order different from the original one. Such packet reordering events could cause serious performance degradation in TCP and UDP applications. As a result, a number of measurement methods have recently been proposed to enable any Internet host to detect packet reordering from itself to another host. However, these methods have encountered a number of practical difficulties, such as rate-limiting and filtering imposed on ICMP and TCP SYN packets. Moreover, some of the methods cannot detect packet reordering in all scenarios. In this paper we present three new methods for end-to-end packet reordering measurement. Since these methods are based on the TCP data channel, the probing and response messages will not be affected by any intermediaries on an Internet path. We have validated and tested the methods in 20 most common systems and implemented them in a tool called POINTER. We also present measurement results obtained from 200 websites in the Internet.

1 Introduction

Packet reordering, which was first identified in the pioneering work of Paxson [1], is still a common phenomenon in the Internet today [2, 3]. The packet reordering is the result of parallelism in network components, e.g., routers and switches, route instability, and load balancing mechanisms [2]. It is well known that packet reordering can adversely affect the performance of TCP and UDP based applications [4, 5, 6]. For example, a TCP flow may enter the fast retransmit state or timeout state if it misinterprets out-of-order delivery as packet losses [4]. To alleviate the effect on the TCP throughput, several proposals have been made to make TCP more robust to packet reordering events [7, 8, 9, 10, 11]. Furthermore, malicious packet reordering can be used to launched Denial-of-Service attacks [12].

Therefore, it is important to measure packet reordering [17] and to quantify the degree of reordering [13, 14]. Previous works have studied various characteristics of packet reordering, such as its frequency, magnitude, and its relationship with other factors. They can be categorized into two main classes—*passive measurement* [15] and *active measurement* [1, 2, 3, 5, 6, 16, 17, 18]. In this paper, we concentrate on the active measurement approaches, which can be further classified into two groups. The first is referred to as *bulk-transport based measurement* that exchanges a relatively large amount of real application traffic between many participating sites [1, 3, 5, 6, 16]. The second is *packet-train based measurement* that sends a train of probing packets, and the feedback information is then used to infer the presence of packet reordering [2, 17, 18].

The main advantage of the bulk-transport based measurement is their ability to measure the impact of packet reordering on real applications and on different transport protocols. Moreover, they can observe the packet reordering phenomena in both the forward-path and backward-path by examining traffic traces in different directions. However, these approaches require coordinations among various participating sites, which obviously cannot be done easily for an arbitrarily large network scale. The packet-train based measurement, on the other hand, allows any Internet host to measure packet reordering on the paths between itself and any other host. Moreover, this approach usually requires only a small number of packets to conduct the measurement.

The focus of this paper is on the packet-train based measurement methods, which unfortunately still suffer from a number of practical limitations. First, the ICMP and TCP packets used in the ICMP-based approach [2] and the TCP SYN Test [17] are often rate-limited or even filtered by firewalls [19]. Second, both the Dual Connection Test [17] and Tulip [18] rely on the assumption that the ID field in the IP header (IPID) increases monotonically across TCP connections for the same server. This assumption, unfortunately, does not hold in some popular systems, such as Linux and OpenBSD. Third, the Single Connection Test [17] may yield inaccurate results due to the delayed acknowledgment algorithm. The Dual Connection Test also suffers from inaccurate results in the presence of load balancing schemes. Fourth, both the TCP Data Transfer Test [17] and the ICMP-based method [2] cannot distinguish all packet reordering scenarios. Moreover, all the aforementioned methods detect *end-to-end* packet reordering. That is, two packets can arrive inorder even though they may be reordered in an even number of times on the path.

In this paper we propose three new methods to end-to-end packet reordering measurement, which can detect all four reordering cases: *no-reordering*, *forward-path reordering*, *backward-path reordering*, and *dual-path reordering*. Since they use TCP data as probing messages in a single connection, the probing messages and the responses will not be filtered by firewalls or routers. Moreover, each measurement is conducted in one TCP session; therefore, the measurement result will not be affected by a content-blind load balancing scheme. To reduce the impact on the normal network traffic, each method injects only a minimal number of packets into the network, and the size of the response packets is also small.

The three methods differ from each other only in the mechanisms to trigger the required responses from the remote host, catering for the diverse TCP implementations in the Internet. Another important feature of our methods is that they only rely on the TCP sliding window protocol, which is supported by all TCP variants (e.g. TCP Tahoe, TCP Reno, TCP NewReno, etc.) [21, 22]. Moreover, our approaches make use of customized receiving window size and MSS (Maximum Segment Size) to control the number and size of response packets. Furthermore, we have employed a timeout mechanism to discard suspicious samples as a result of packet losses. We have implemented the three methods in a tool called POINTER, and have thoroughly tested and verified the approaches in a test-bed and 200 websites.

The rest of the paper is organized as follows. In section 2, we introduce the principles behind the three new approaches, including a short review on the TCP sliding window protocol. Based on the general approach, we detail in section 3 the three measurement methods. In section 4, we present the measurement results obtained from a test-bed and the Internet for the validation of the methods. Moreover, we have analyzed the correlation of packet reordering events on a backward path from two websites, and discovered that they share the same path where packet ordering occurs. We finally conclude this paper in section 5 with current work.

2 The fundamental principles of the new approaches

In the next subsection we first discuss some general fundamental principles for the design of a reordering detection scheme. The results presented there will then lead to a general requirement for designing a comprehensive and effective scheme to detect packet reordering. After that, we sketch our approach of meeting this requirement based on the TCP sliding window mechanism. In the rest of this paper, we refer the host that performs packet reordering measurement to a *client*, and the host on the other side of the TCP connection to a *server*.

2.1 The fundamental principles

Proposition 1. *Assume that during each probing session a client can trigger k distinguishable responses out of a set of n possible ones from a server. Then, n must be at least 3 in order to discriminate all the 4 packet reordering scenarios.*

Proof. Note that the k responses received from the server are *ordered* according to their times of generation. Therefore, the client can trigger a set of $P(n,k) = n!/(n-k)!$ k-response. In order to differentiate the 4 reordering scenarios, clearly $n \geq 3$, because $P(2,k) < 4$, and $P(n,k) > 4$ for some $k > 0$ when $n \geq 3$. □

Proposition 2. *Given that the smallest n is used, the minimum number of distinguishable responses from a server required for a complete reordering detection during each probing session is 2. Therefore, if one probing message will trigger at least one distinguishable response, then the client only needs to send 2 probing messages during each session.*

Proof. There are two possible cases for $n = 3$: $k = 2, 3$ for which $P(3,2) = P(3,3) = 6 > 4$. □

Corollary 1. *Consider that a client sends 2 probing messages—$M1$ first and then $M2$—to trigger a generation of 2 normal, distinguishable responses from a server, denoted by $R1$ and $R2$. Then, the scheme can distinguish all 4 reordering cases if $M1$ and $M2$ can trigger another distinguishable response $R3$, and, by definition, $R3$ is not a normal response.*

Proof. This is a direct consequence of Proposition 2, which requires one more distinguishable response for a complete reordering detection. □

The result of Corollary 1 gives us a hint in designing a comprehensive and effective reordering detection scheme. First of all, consider that the server responds with $R1$ and $R2$ upon receiving $M1$ and $M2$ in order. That is, if there is no forward-path reordering, $R1$ will be generated first

Table 1: Variables used in the TCP sliding window mechanism.

Name	Description
`SND.UNA`	Oldest unacknowledged sequence number (SN)
`SND.NXT`	SN of the next segment to be sent
`SND.WND`	Size of the send window
`RCV.NXT`	SN of the next segment to be received
`RCV.WND`	Size of the receive window
`SEG.ACK`	Acknowledgment number (AN) of a segment
`SEG.SEQ`	First SN of a segment
`SEG.LEN`	The length of a segment in bytes

and then followed by $R2$. Therefore, the order of receiving $R1$ and $R2$ on the client side can differentiate between the cases of no-ordering and backward-ordering.

To trigger the third response $R3$, we need an "erroneous event" taken place at the server. The novelty here is that we use the reordering of $M1$ and $M2$ to serve as the erroneous event. That is, if there is forward-path reordering, the receiver will be compelled to send $R3$ and another normal response ($R1$ or $R2$). For instance, the receiver responds with $R3$ first and then $R1$. Then the order of receiving them by the client can differentiate between the cases of forwarding-reordering and dual-path reordering.

Based on the discussion above, we only need a pair of probing messages $M1$ and $M2$ to detect all four reordering cases. Note that we have so far assumed no packet losses and a reasonable amount of latency between the message arrivals at the server. Moreover, the server's responses to $M1$ and $M2$ only depend on the information inscribed in the messages and the relative order of the two messages received by the server. We will address the effect of packet losses and the solution to them in section 3.4. In the next section, we first present several possibilities of selecting $M1$ and $M2$ based on the TCP sliding window mechanism.

2.2 The TCP-based probing message pair

Since our methods are based on the TCP data channel, we first review the TCP sliding window algorithm. We adopt the notations in Table 1 [20] for the following discussion. A TCP sender uses a *send window* to control the transmission of segments, while a TCP receiver maintains a *receive window* for receiving segments from the sender. When a TCP receiver is in the ESTABLISHED state and a segment arrives, it will process the segment according to the following order [20]: (1) check the SN, (2) check the `RST` bit, (3) check security and precedence, (4) check the `SYN` bit, (5) check the AN, (6) check the `URG` bit, (7) process the segment text, and (8) check the `FIN` bit.

In our proposed methods, a client sends a probing message pair to induce two TCP data segments from the server when there is no packet reordering on the forward path, i.e., the probing message pair passes all eight steps. On the other hand, the reordering of the probing message pair, which is perceived as an erroneous event at the server, will induce the transmission of a pure ACK and a data segment. There are altogether three ways of generating the pure ACK.

First of all, a pure ACK can be generated as a result of not passing step 1 where a TCP receiver performs a *sequence number check* (SNC). The purpose of the SNC is to determines whether the received segment's SN is acceptable according to Eq. (1) or Eq. (2). If the segment fails the SNC (an erroneous event), some TCP implementations will drop the segment and return a pure ACK whose value is specified in the second part of Eq. (3).

A pure ACK can also be generated as a result of not passing step 5, where a TCP receiver performs an *acknowledgment number check* (ANC) based on Eq. (4). If the ACK is acceptable, the receiver will update its send window by setting `SND.UNA` = `SEG.ACK`. Otherwise, if `SEG.ACK` < `SND.UNA`, i.e., a duplicate ACK, the receiver will ignore it. Moreover, if `SEG.ACK` > `SND.NXT` (an erroneous event), i.e., acknowledging bytes that have not been sent, some TCP implementations will drop the segment and send back a pure ACK whose value is specified in the second part of Eq. (3).

Finally, if a TCP implementation does not respond to both failed SNC and failed ANC, a pure ACK can still be induced by sending a out-of-ordered segment. Accordingly, we have designed three different probing message pairs based on the server's responses discussed above. The corresponding methods are referred to as ACM (*ACknowledgment based Measurement*) for the response to a failed SNC, $SAM1$ (*Sequence number and ACK based Measurement*) for the response to a failed ANC, and $SAM2$ for the response to a out-of-order segment.

$$\texttt{RCV.NXT} \le \texttt{SEG.SEQ} < \texttt{RCV.NXT} + \texttt{RCV.WND}. \tag{1}$$

$$\texttt{RCV.NXT} \le \texttt{SEG.SEQ} + \texttt{SEG.LEN} - 1 < \texttt{RCV.NXT} + \texttt{RCV.WND}. \tag{2}$$

$$\texttt{SEG.SEQ} = \texttt{SND.NXT},\ \texttt{SEG.ACK} = \texttt{RCV.NXT}. \tag{3}$$

$$\texttt{SND.UNA} < \texttt{SEG.ACK} \le \texttt{SND.NXT}. \tag{4}$$

We have tested 20 common systems to observe their responses to the failed SNC and failed ANC. As shown in Table 2, the majority of the tested systems responded to unacceptable ACKs. Therefore, the ACM method can be applied to detect packet reordering on the paths to these systems. Although the Linux systems do not respond to unacceptable ACKs, they respond to unacceptable SNs. Thus, the SAM1 method can be applied to them. Finally, the SAM2 method can be applied to the VM and HP-UX systems which ignore both unacceptable ACKs and unacceptable SNs.

Table 2: Popular operating systems and the new measurement methods.

Operating systems	Response to unacceptable SNs	Response to unacceptable ACKs	Measurement Methods
NT4/Win98, Win2000, WinXP, Win2003, MaxOSX, NetWare, SCO UNIX, NetBSD, AIX, OS/2, IRIX, Tru64, FreeBSD, Solaris 9 , Solaris 8, Solaris*, OpenVMS	Not tested	Send a pure ACK	ACM
Linux	Send a pure ACK	No response	SAM1
VM and HP-UX	No response	No response	SAM2

3 Three new measurement methods

The three measurement methods use different pairs of TCP data segments to induce from the server two data segments in the absence of reordering in the forward path, and one pure ACK and one data segment in the presence of reordering in the forward path. Moreover, they do not assume any specific TCP-based application running at the server. The only assumption is that the server replies with at least several hundreds of bytes of data in response to the client's probing messages, which is clearly possible for most popular TCP-based services. For example, the HTTP `GET` can be used to fetch a sufficiently large web object from a web server. The `GET` command in most FTP clients can be used to download a suitable file from a public FTP server. In the following, we illustrate the three methods using HTTP, but they can be easily adapted for FTP, POP3, or other TCP applications.

The probing session can be carried out at any time in the ESTABLISHED state. To keep the following discussion simple, however, we start the probing procedure immediately after the TCP three-way handshake is completed. The notations used in the remaining of this paper are summarized in Table 3, and the segments involved in the 3-way handshaking are given in Table 4. Note that these segment exchanges are the same for all three methods. C_0 and S_0 are the initial SNs for the client and server, respectively. In $TC1$, the client initiates a SYN segment with a *small* advertised window size of W_C. The main function of the small window size is to ensure that the server will adopt W_C for its send window size. Therefore, even before issuing the HTTP request in $TC2$, the client is able to predict correctly that the server will return one data segment and the size of the TCP payload (W_C), provided that the size of the requested document is larger than W_C. The packet sequence concerned is shown in (3)-(5) in Table 4. As we shall see later, the ANs in the probing message pair are the same for all three methods, because they all exploit the predicability of the server's payload through the small W_C. However, they differ in their SNs and the TCP payloads.

3.1 The ACM method

Recall that the *ACM* method is based on the server's response to the recipient of an unacceptable ACK. In this case, the server is expected to drop the corresponding segment and to respond with an ACK. Figs. 1(a) and 1(c) first show the packet sequences in the absence of packet reordering in the forward path. The probing message pair used in this method are two back-to-back, pure ACKs ($TC3$ and $TC4$). Notice from Table 5 that $TC3$ acknowledges the receipt of $DataS0$, whereas $TC4$ acknowledges a "yet-to-receive" data segment. Because of the predicability of the payload length in $DataS1$, the client is able to send $TC4$ with a correct AN immediately after $TC3$.

If these two ACKs arrive at the server in order (Figs. 1(a) and 1(c)), the server will expectedly transmit two data segments $TS4$ and $TS5$, one after the other, which contain the requested document. The order that $TS4$ and $TS5$ are received at the client can then be used to differentiate between the no-ordering and backward-path ordering cases.

However, if these two ACKs are reordered in the forward path (Figs. 1(b) and 1(d)), the server will discover that $TC4$ is an unacceptable ACK, because $\texttt{SND.UNA} = S_1$ and $\texttt{SND.NXT} = S_1 + W_C$ at the time of receiving the ACK. In this case, the server, which is implemented under the first group of systems in Table 2, is expected to drop $TC4$ and to send back a pure ACK $TS4$. When $TC3$, the first ACK, later arrives, the server will send out the requested data in $TS5$. Thus, the order of receiving $TS4$ and $TS5$ can be used to differentiate between the forward-path and dual-path reordering cases.

To ensure a proper working of this method, the size of the requested document must be at least $3 \times W_C$ bytes. Since the probing message pair used in the ACM method are pure ACKs, the active probings have a minimal impact on the normal network traffic. Furthermore, the server's response involves at most $3 \times W_C$ bytes of data ($DataS0$, $DataS1$ and $DataS2$) for differentiating between the no-reordering and backward-path reordering cases. For the other two cases, the amount of data required is even less: $2 \times W_C$ bytes ($DataS0$ and $DataS1$).

Table 3: Notations used in the description of the new measurement methods.

Notations	Description
$TCi, (i = 1, \ldots, 4)$	The ith segment dispatched by the client
$TSi, (i = 1, \ldots, 5)$	The ith segment dispatched by the server
SYN	The TCP SYN packet sent by the client
$SYN\text{-}ACK$	The TCP SYN/ACK packet responded by the server
$CmdCi, (i = 1, 2, 3)$	ith HTTP request sent by the client
$X_i, (i = 1, 2, 3)$	Size of $CmdCi$ in bytes
$ACKCi, (i = 1, 2)$	ith pure ACK sent by the client in the ESTABLISHED state
$ACKSi, (i = 0, 1, 2)$	ith pure ACK sent by the server in the ESTABLISHED state
$DataSi, (i = 0, 1, 2)$	ith data segment containing the HTTP response sent by the server
W_C	The size of the client's advertised window
W_S	The size of the server's advertised window

Table 4: The packet sequence common for all three methods.

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
1	$TC1$	C_0	0	SYN	0
2	$TS1$	S_0	C_1=C_0+1	SYN/ACK	0
3	$TC2$	C_1	S_1=S_0+1	HTTP request ($CmdC1$)	X_1
4	$TS2$	S_1	C_2=C_1 + X_1	Pure ACK ($ACKS0$)	0
5	$TS3$	S_1	C_2	HTTP reply ($DataS0$)	W_C

3.2 The SAM1 method

The second method is based on the server's response to unacceptable SNs which can be applied to Linux systems. In this case, a Linux server drops the corresponding segment and responds with an ACK. Figs. 2(a) and 2(c) show the packet traces when there is no packet reordering in the forward path. The probing message pair in this case are $TC3$ and $TC4$ which are a second HTTP request and a pure ACK, respectively. Similar to the case for the ACM method, $TC4$ acknowledges a "yet-to-receive" data segment from the server. The purpose of sending the second HTTP request message, on the other hand, will be clear from the following explanation.

From the parameters in Table 6, it is not difficult to see that the server will send two more data segments, each in the size of W_C, in response to the probing message pair when there is no forward-path reordering. Therefore, the order of receiving the two data segments can be used to differentiate between the no-ordering and backward-path reordering cases.

On the other hand, if the probing message pair is reordered (Figs. 2(b) and 2(d)), $TC4$ will fail the SNC on the server side, because the SN falls outside the legal range according to Eq. (2). Specifically, from Table 6, $TC4$'s SN is given by $\texttt{SEG.SEQ} = C_2 + X_2 + W_S - 1$ ($X_2 > 1$) and $\texttt{SEG.LEN} = 0$ (0 payload length). On the other hand, at the time of receiving $TC4$, the server's states as a TCP receiver are given by $\texttt{RCV.NXT} = C_2$ and $\texttt{RCV.WND} = W_S$. According to Eq. (2), $TC4$'s SN is therefore illegal, i.e., $\texttt{SEG.SEQ} > \texttt{RCV.NXT} + \texttt{RCV.WND}$. As discussed before, the server is therefore expected to discard this pure ACK and to respond with an ACK ($ACKS1$). When $TC3$ that contains a correct SN arrives, the server will clearly respond with $DataS1$. Therefore, the order of receiving $ACKS1$ and $DataS1$ can differentiate between the backward-path reordering and dual-path reordering cases. Thus, unlike the ACM method that sends out two pure ACKs, this method sends a second HTTP request, so that the TC4's SN is different from TC3's.

Same as the ACM method, the total size of the requested document must be at least $3 \times W_C$ to ensure a proper working of this method. In terms of the overhead, the size of the probing message pair for the $SAM1$ method is larger than that in the ACM method by X_2 bytes, the size of the second HTTP request message. However, the maximum amount of HTTP data returned by the server is the same for both methods, i.e., $3 \times W_C$ bytes.

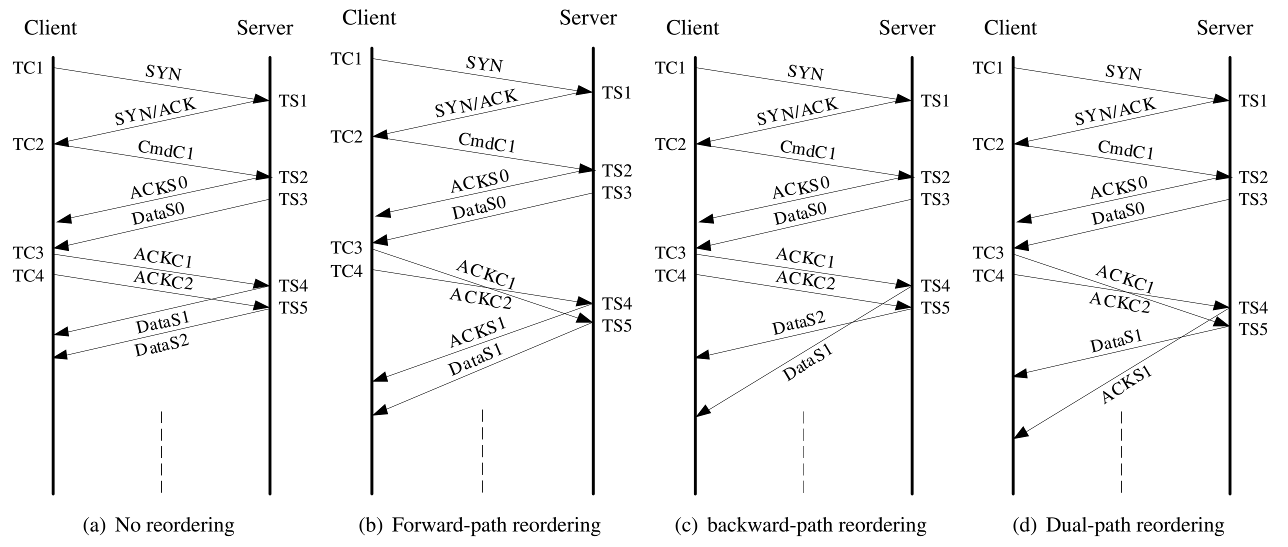


Figure 1: Packet sequences for the ACM method under the four packet reordering scenarios.

Table 5: The packet sequence for the ACM method (continued from Table 4).

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	C_2	$S_2 = S_1 + W_C$	Pure ACK ($ACKC1$)	0
7	$TC4$	C_2	$S_3 = S_1 + 2W_C$	Pure ACK ($ACKC2$)	0
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	C_2	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	C_2	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_2	C_2	Pure ACK ($ACKS1$)	0
9'	$TS5$	S_2	C_2	HTTP reply ($DataS1$)	W_C

3.3 The SAM2 method

Recall from the beginning of this section that the TCP implemented in HP-UX and VM do not respond to both failed SNC and failed ANC. The $SAM2$ method described in this section is designed to cater for this set of TCP implementations. Figs. 3(a) and 3(c) illustrate the packet sequences for this method when there is no packet ordering in the forward path. After receiving the first data segment $TS3$, the client sends out the probing message pair $TC3$ and $TC4$.

Similar to the $SAM1$ method, the probing message pair uses different SNs. However, notice from Table 7 that the SNs are offset by Off and $2 \times Off$ respectively, where Off is a small value. That is, both messages contain unexpected but *acceptable SN* (out-of-order segments). Therefore, the recipient of $TC3$ will enable the server to advance its send window and to send a data segment. When $TC4$ later reaches the server, this segment acknowledges all the outstanding data sent by the server (AN = SND.NXT $= S_1 + 2W_C$); therefore, the server will also advance its send window. As a result, the server will reply with another data segment $TS5$. The server therefore responds with a total of two data segments whose order of arriving at the client can be used to differentiate between the no-ordering and backward-path reordering cases.

If the two messages are reordered, as shown in Figs. 3(b) and 3(d), the message $TC4$ contains an *unacceptable ACK*. However, the HP-UX and VM server will ignore the illegal AN, but, similar to other TCP implementations, it will definitely respond to the out-of-ordered segment with a duplicate ACK, $TS4$. There is a minor difference between the HP-UX and VM systems though. $TS4$ contains a SN of S_1 for an HP-UX server) whereas $TS4$ contains a SN of $S_1 + W_C$ for a VM server, and both contain an AN of C_2. However, this difference does not affect our measure-

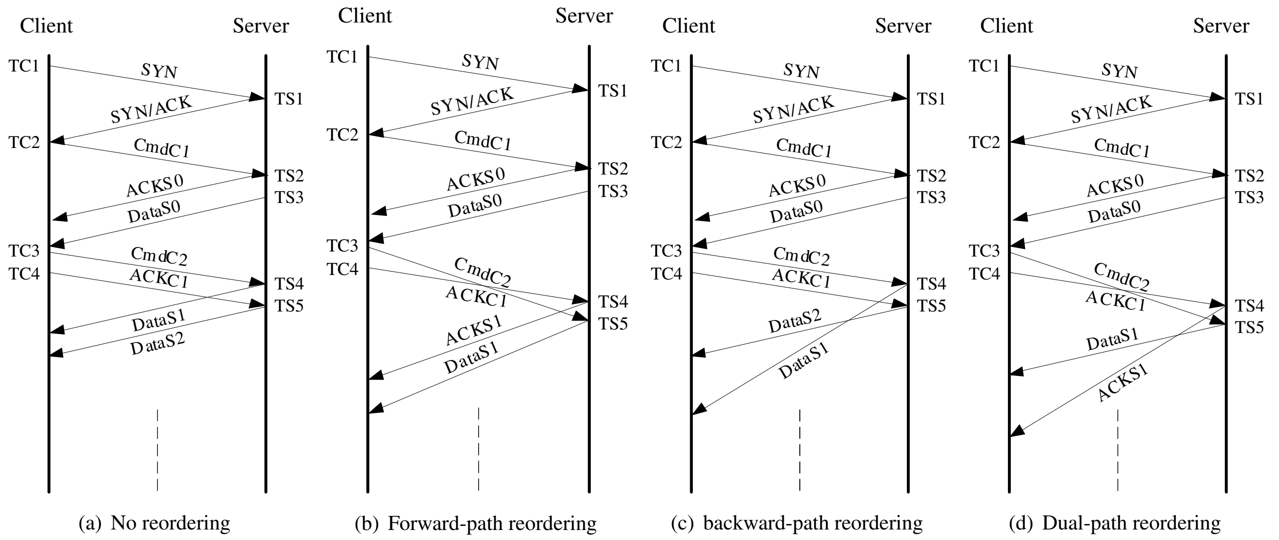


Figure 2: Packet sequences for the $SAM1$ method under the four packet reordering scenarios.

Table 6: The packet sequence for the $SAM1$ method (continued from Table 4).

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	C_2	$S_2 = S_1 + W_C$	HTTP request ($CmdC2$)	X_2
7	$TC4$	$C_2 + X_2 + W_S - 1$	$S_3 = S_1 + 2W_C$	Pure ACK ($ACKC1$)	0
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	$C_2 + X_2$	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	$C_2 + X_2$	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_2	C_2	Pure ACK ($ACKS1$)	0
9'	$TS5$	S_2	$C_2 + X_2$	HTTP reply ($DataS1$)	W_C

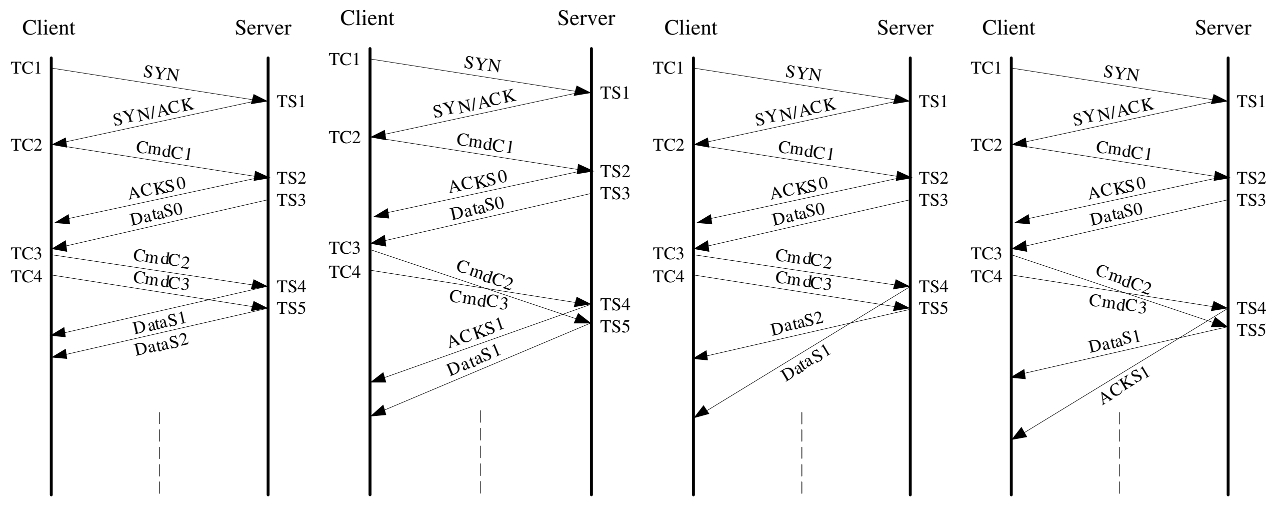
ment method, and in fact it can be used to remotely identify these two systems [19]. On the other hand, the second message $TC3$ contains an *acceptable* AN of $S_1 + W_C$. In this case, the server will advance its `SND.UNA` and reply with a data segment $TS5$ with $SN = S_1 + W_C$ and $AN = C_2$. Therefore, the order of receiving the ACK and data segment from the server can be used to differentiate between the backward-path reordering and dual-path reordering cases.

The size of the requested document must again be at least $3 \times W_C$ to ensure a proper working of the method. It is not difficult to show that the maximum amount of data required by the $SAM2$ method is the same as that for the $SAM1$ method, which is only $3 \times W_C$ bytes. The size of the probing packets in the $SAM2$ method is larger than that of the ACM method by the size of $CmdC3$.

3.4 Packet losses control

Packet losses in the probing packets and responses will clearly affect the measurement results. Some of them can be easily detected by observing abnormal responses from the server, and they merely delay the observation period. On the other hand, other kinds of packet losses may introduce a bias in the measurement results, such as when the responses are lost and then retransmitted. Due to the limited space, we only describe the solutions for the ACM method under the second type of packet losses, which can be easily modified for the $SAM1$ and $SAM2$ methods.

To illustrate the problem, consider the scenario in Fig. 4(a) where there is backward-path reordering and the segment $TS5$ is lost. If the client interprets the segment $TS4$ and the retransmitted segment $TS5$ as the server's responses to its probing messages, it will arrive at a false conclusion that there is no packet reordering.



(a) No reordering (b) Forward-path reordering (c) backward-path reordering (d) Dual-path reordering

Figure 3: Packet sequences for the $SAM2$ method under the four packet reordering scenarios.

Table 7: The packet sequence for the $SAM2$ method (continued from Table 4).

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	$C_2 + Off$	$S_2 = S_1 + W_C$	HTTP request ($CmdC2$)	X_2
7	$TC4$	$C_2 + 2Off$	$S_3 = S_1 + 2W_C$	HTTP request ($CmdC3$)	X_3
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	C_2	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	C_2	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_1 *or* $(S_1 + W_C)$	C_2	Pure and duplicate ACK ($ACKS1$)	0
9'	$TS5$	S_2	C_2	HTTP reply ($DataS1$)	W_C

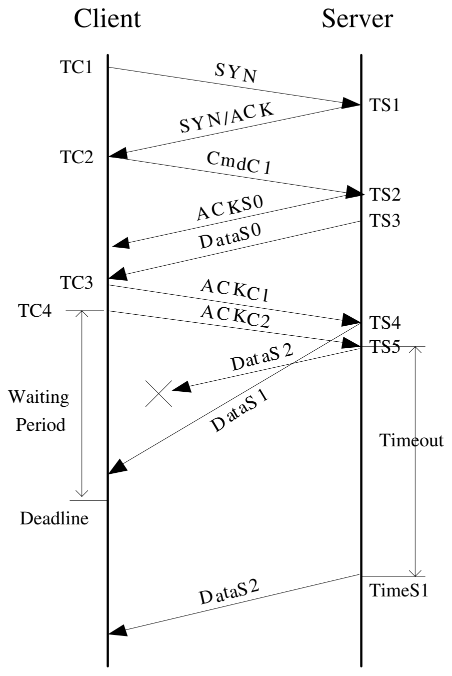
To remedy this problem, we impose a deadline for receiving responses from the server. As illustrated in Fig. 4(a), after the client sends $TC4$, it will not accept any response arriving after the deadline and declare the measurement unsuccessful. In another example, Fig. 4(b) depicts that the dual-path reordering would be mistaken for the forward-path reordering if the deadline mechanism were not used. It is obvious that the server can only use the timeout mechanism to retransmit the lost packet because it is impossible to trigger the fast retransmit mechanism. We therefore set the deadline to $1.5 \times \overline{RTT}$, where $\overline{RTT}$ is the mean value of RTT between the client and the server.

4 POINTER and the measurement results

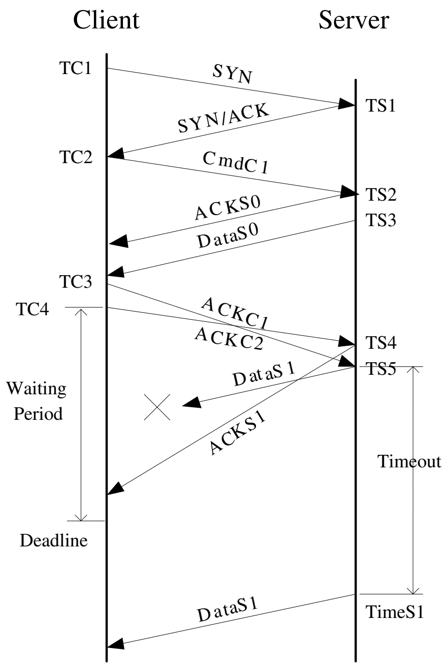
We have developed a measurement tool called POINTER (*Packet reOrderINg tesTER*) that implements the ACM, SAM1, and SAM2 methods. The implementation uses the packet filter API available from [27] to block the TCP RST packets generated by the local host, and the WinPcap library [28] to generate customized TCP packets. With POINTER, we can validate the three measurement methods on a test-bed and in the Internet, and analyze the packet reordering statistics.

4.1 Measurement results from a test-bed

We have validated the ACM and $SAM1$ methods on a test-bed environment. As shown in Fig. 5, the test-bed consists of two servers, a POINTER client, and an Iperf host. The Iperf client is responsible for generating background traffic. Apache web servers are running on both the Linux server (Linux 2.4.20-8) and Windows 2000 server for validating the SAM1 method and the ACM method, respec-



(a) A packet loss in the presence of backward-path reordering



(b) A packet loss in the presence of dual-path reordering

Figure 4: Handling packet losses in the ACM method.

tively. Both the servers and the POINTER client use tcpdump or Windump to capture the probing packets and response packets. The servers and clients are connected by a FreeBSD-based router running Dummynet [31] to simulate multipath between a client and a server.

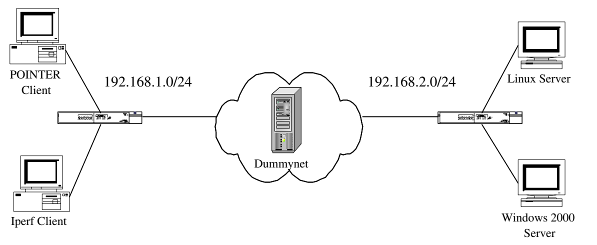


Figure 5: The network configuration of the test-bed.

By setting 20 pipes between the subset of 192.168.1.0/24 and 192.168.2.0/24 in the Dummynet, we can emulate the four packet reordering scenarios on the test-bed. We let each method keep on probing the server until it has detected 100 packet reordering events in the probing packet pairs. The measurement results are then compared with the packet traces. Our findings show that all the detection results produced by the ACM and SAM1 methods are correct for all test cases. Moreover, we have validated the two methods on other systems available in our department, including WinXP, Netware, FreeBSD, and Solaris. To validate the methods with the remaining systems listed in Table 2, we have conducted the experiments with the WWW servers in the Internet, to be presented next.

4.2 Measurement results from the Internet

We have made used of the service provided by NetCraft to identify the systems running in the WWW servers [29]. We have also obtained additional information, e.g. from [30], to ensure correct system identification. We have altogether selected around 200 websites, part of which are randomly chosen from Yahoo random URL database [32] suggested by [17] and part of which include the popular websites, e.g. Microsoft, Apple, Hotmail, Yahoo, Google, etc. For more popular systems, such as Linux and Windows, we have tested 10 servers for each system. However, for OS/2 and OpenVMS, we could just locate a handful of websites, such as `www.os2.org` and `www.openvms.compaq.com`. For both Solaris 2 and Solaris 7, NetCraft has identified them to be the same, and labelled them by Solaris. Therefore, our tests may not have covered both Solaris 2 and Solaris 7, which are marked with $*$ in Table 2.

For each web server, the validation process consists the following steps:

1. carry out the measurement process in no reordering case, i.e. sending out the segments $TC3$ and then

$TC4$, and recording the responses $TS4$ and $TS5$ from the server.

2. carry out the measurement process in forward reordering case, i.e. sending out the segments $TC4$ and then $TC3$, and recording the responses $TS4'$ and $TS5'$ from the server.

3. If the responses, $\{TS4, TS5, TS4', TS5'\}$ fulfill the requirement of Corollary 1, then the packet reordering in the path from the monitoring point to remote host can be measured.

The measurement results obtained from these websites have confirmed the correctness of the three methods and produced the measurement results in Table 2. For an arbitrary accessible web server, we do not need to first identify the OS type of remote host before measuring packet reordering. We can conduct the validation process of ACM, SAM1 and SAM2 sequently to find out which method is proper.

Moreover, we have conducted more measurements from 100 WWW servers among the 200 websites used in the above validation process, and obtained 500 measurement results for each server. Fig. 6 shows the empirical cumulative distribution function (CDF) of the forward-path reordering rate and backward-path reordering rate. The measured dual-path reordering rate is zero for all cases. The rate is given by the percentage of measurements that indicate the presence of packet reordering. Then we rank all the servers according to a nondecreasing order of their reordering rates, and obtain the CDF. The results show that more than 35% of the paths experienced forward-path reordering at least once, and backward-path reordering was observed on about 10% of the paths. Moreover, the forward-path reordering was more prevalent than the backward-path reordering in terms of the percentage of reordering events. The forward-path reordering rate could go up as high as 0.4, while that for the backward-reordering was only 0.3.

4.3 Correlation of packet reordering events

In this section we analyze the correlation of packet reordering events. To this end, we have conducted measurements on `www.applecomputer.com` and `kidsafe.apple.com` over a 2-month period. These two sites have been found to have more than 10% forward-path reordering rate according to [17]. In each measurement, the client first sends out the probing pair to `www.applecomputer.com` and then another probing pair to `kidsafe.apple.com`. The time gap between sending the two probing pairs is very short. There is a random delay (between 1 second and 2 seconds) between two consecutive measurements. We have conducted 59 sets of measurements and each set consists of 100 samples.

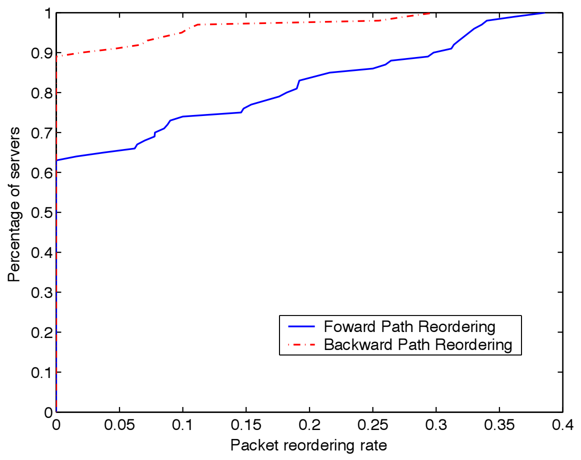


Figure 6: The empirical cumulative distribution functions of the packet reordering rates.

Unlike the previous results for the 100 websites, the results here indicate that there are more backward-path reordering events than the forward-path ones. Fig. 7(a) shows the mean backward-path reordering rates to both servers obtained from the 59 sets of measurement. Each point is an average of the 100 samples within a set of measurement. The rates for both servers are very similar in many sets of samples. Moreover, the overall means rates for both servers are approximately equal to 16.7%. To probe into the issue further, we compute the pair-difference test statistic for the 59 sets of data [33], and find that with a 99% confidence interval the backward-path reordering rates are similar for the two servers.

Next, we study and compare the time series of the measurements obtained for the two servers. We use $TR_{i,j}^{S_k}$, $i = 1, \ldots, 59$, $j = 1, \ldots, 100$, $k = 1, 2$, to denote the presence/absence of backward-path reordering in the corresponding sample. S_1 refers to `www.applecomputer.com` and S_2 refers to `kidsafe.apple.com`. $TR_{i,j}^{S_k} = 1$ in the presence of backward-path reordering; otherwise, $TR_{i,j}^{S_k} = 0$. We compute the autocorrelation value for the ith set of data according to [34]

$$\sum_{j=1}^{99-m} TR_{i,j}^{S_k} TR_{i,j+m}^{S_k}, \; m = 0, \ldots, 99.$$

We then compute an average of the 59 autocorrelation values for each m and the results are depicted in Fig. 7(b). They first show that the correlation between the reordering events occurring at different times is not strong. Moreover, the two autocorrelation plots are so similar that it is very difficult to distinguish from each other.

Besides the autocorrelation function, we define a *pair-*

reorder index (PI) by

$$PI(i) = \frac{\sum_{j=1}^{100}(TR_{i,j}^{S_1} \,\&\, TR_{i,j}^{S_2})}{(\sum_{j=1}^{100}(TR_{i,j}^{S_1} \mid TR_{i,j}^{S_2}))/2},\ i = 1, \ldots, 59,$$

to measure the likelihood that two packet reordering events would occur within a measurement. The notations $\&$ and $|$ refer to the logical AND and logical OR operators, respectively. Therefore, the higher the index is, the higher the likelihood that backward-path reordering would occur in the paths from both servers to the POINTER client. Fig. 7(c) depicts the $PI(i)$ values computed from the 59 sets of experiments. The mean value of $PI(i)$ over the 59 sets is 15.5%, and 52 sets have nonzero $PI(i)$ values. This result, together with the results presented in Fig. 7(a) and Fig. 7(b), strongly suggest that the two servers shared the same part of the path to the POINTER client, where packet reordering occurred. Moreover, these three sets of statistical analysis can be applied to study the correlation of other Internet path statistics.

5 Conclusions and current work

In this paper we have presented three novel methods for end-to-end packet reordering measurement—ACM, $SAM1$ and $SAM2$. Unlike the previous approaches, we have designed the probing messages based on the TCP data channel, thus solving the practical problems of going through routers and other intermediaries on the Internet paths. Moreover, the probing message pair is carefully crafted, so that the client can predetermine the returned responses which can be used to confirm whether there is packet reordering on the forward path. Moreover, the order of the arrival of the two response packets is used to confirm whether there is packet reordering on the backward path. Thus, the methods can detect all four packet reordering scenarios. The amount of data used is also kept to a minimum.

We have implemented the three methods in POINTER and validated the methods in 20 most common systems in both a test-bed and the Internet. We are now in the process of making the tool available in different platforms. With POINTER, one can detect packet reordering on any path in the Internet. Moreover, the tool can be used to study other important issues, such as the correlation of packet reordering events presented in this paper. Furthermore, we are in the process of improving these three methods and conducting a much larger-scale measurement study on the prevalence of packet reordering in the Internet today.

Acknowledgments

The work described in this paper was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 5080/02E) and a grant from the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99). We thank Mr. Kent Leung and his colleagues in the ITS office for their help in conducting the Internet measurement. We also thank the anonymous reviewers for their useful comments.

References

[1] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3), June 1999.

[2] C. Partridge, J. Bennett, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6), December 1999.

[3] L. Gharai, C. Perkins, and T. Lehman. Packet Reordering, High Speed Networks and Transport Protocol Performance. In *Proc. IEEE ICCCN*, October 2004.

[4] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, 7(6), September 2002.

[5] D. Loguinov and H. Radha. Measurement study of low-bitrate internet video streaming. In *Proc. ACM Internet Measurement Workshop*, November 2001.

[6] X. Zhou and P. Mieghem. Reordering of IP packets in Internet. In *Proc. Passive and Active Measurement*, April 2004.

[7] M. Allman and E. Blanton. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1), January 2002.

[8] S. Floyd, M. Zhang, B. Karp, and L. Peterson. RR-TCP: A reordering-robust tcp with DSACK. In *Proc. IEEE ICNP*, November 2003.

[9] J. Lee, C. Lim, S. Bohacek, J. Hespanha, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *Proc. IEEE Conf. Distributed Computing Systems*, May 2003.

[10] C. Ma and K. Leung Improving TCP robustness under reordering network environment. In *Proc. IEEE GLOBECOM*, November 2004.

[11] A. Sathiaseelan and T. Radzik Improving the Performance of TCP in the Case of Packet Reordering. In *Proc. IEEE HSNMC*, June 2004

[12] I. Aad, J. Hubaux, and E. Knightly. Denial of Service Resilience in Ad Hoc Networks. In *Proc. ACM MobiCom*, September 2004.

[13] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for IPPM. draft-ietf-ippm-reordering-05.txt, Internet Draft, IETF, 2004.

[14] A. Bare, T. Banka, and A. Jayasumana. Metrics for degree of reordering in packet sequences. In *Proc. IEEE Conf. Local Computer Networks*, November 2002.

[15] C. Diot, J. Kurose, J. Jaiswal, G. Iannaccone, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. In *Proc. IEEE Infocom*, April 2003.

[16] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proc. ACM SIGCOMM*, November 1997.

[17] J. Bellardo and S. Savage. Measuring packet reordering. In *Proc. ACM Internet Measurement Workshop*, November 2002.

[18] D. Wetherall, R. Mahajan, N. Spring, and T. Anderson. User-level Internet path diagnosis. In *Proc. ACM Symp. Operating Systems Principles*, October 2003.

[19] R. Spangler. Analysis of remote active operating system fingerprinting tools. http://www.packetwatch.net, May 2003.

[20] J. Postel. Transmission control protocol. RFC 793, IETF, September 1981.

[21] J. Padhye and S. Floyd. On inferring TCP behavior. In *Proc. ACM SIGCOMM*, August 2001.

[22] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3), July 1996.

[23] V. Paxson and M. Allman. Computing TCPs Retransmission Timer. RFC 2988, IETF, November 2000.

[24] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet http://www.icir.org/tbit, December 2004.

[25] TCP Tunable Parameters. Solaris Reference Manual http://docs.sun.com/app/docs/doc/816-0607/6m735r5g6.

[26] Cross-Referencing Linux. http://lxr.linux.no.

[27] Packet filtering reference, platform sdk. http://msdn.microsoft.com.

[28] Winpcap, the free packet capture architecture for Windows. http://winpcap.polito.it.

[29] Netcraft Ltd. http://uptime.netcraft.com/up/accuracy.html.

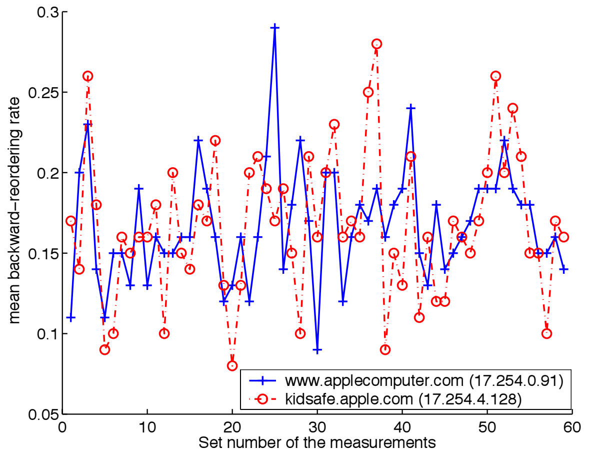
[30] VM/ESA based WWW servers. http://vm.cfsan.fda.gov/vmcms.html.

[31] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, January 1997.

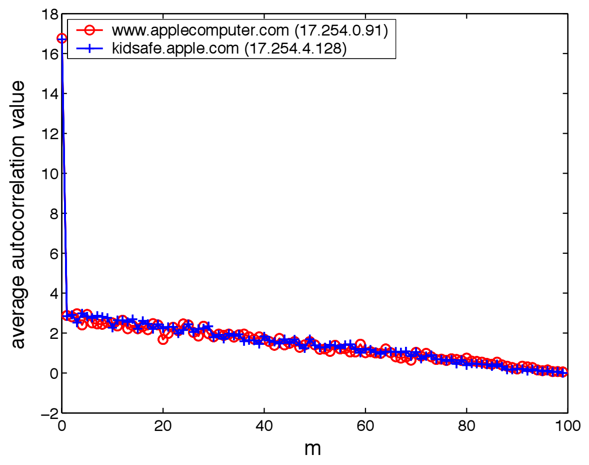
[32] Yahoo! Inc. Random Yahoo! Link. http://random.yahoo.com/bin/ryl.

[33] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

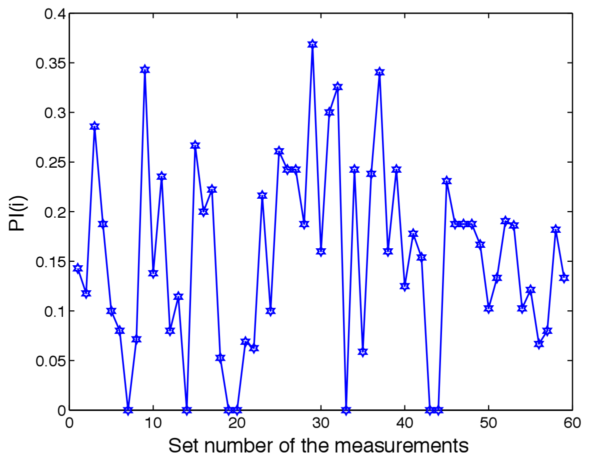
[34] S. Orfanidis. *Optimum Signal Processing. An Introduction. 2nd Edition*. Prentice-Hall, Englewood Cliffs, NJ, 1996.



(a) The mean backward-path reordering rates.



(b) The average autocorrelation of backward-path reordering.



(c) The pair-reorder index for backward-path reordering.

Figure 7: Measurement results for the paths from `www.applecomputer.com` and `kidsafe.apple.com` to the POINTER client.

Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic

Michael Bailey
University of Michigan
mibailey@eecs.umich.edu

Evan Cooke
University of Michigan
emcooke@umich.edu

Farnam Jahanian
University of Michigan
Arbor Networks, Inc.
farnam@umich.edu

Niels Provos
Google, Inc.
provos@google.com

Karl Rosaen
University of Michigan
krosaen@umich.edu

David Watson
University of Michigan
dwatson@eecs.umich.edu

Abstract

Threats to the privacy of users and to the availability of Internet infrastructure are evolving at a tremendous rate. To characterize these emerging threats, researchers must effectively balance monitoring the large number of hosts needed to quickly build confidence in new attacks, while still preserving the detail required to differentiate these attacks. One class of techniques that attempts to achieve this balance involves hybrid systems that combine the scalable monitoring of unused address blocks (or darknets) with forensic honeypots (or honeyfarms). In this paper we examine the properties of individual and distributed darknets to determine the effectiveness of building scalable hybrid systems. We show that individual darknets are dominated by a small number of sources repeating the same actions. This enables source-based techniques to be effective at reducing the number of connections to be evaluated by over 90%. We demonstrate that the dominance of locally targeted attack behavior and the limited life of random scanning hosts result in few of these sources being repeated across darknets. To achieve reductions beyond source-based approaches, we look to source-distribution based methods and expand them to include notions of local and global behavior. We show that this approach is effective at reducing the number of events by deploying it in 30 production networks during early 2005. Each of the identified events during this period represented a major globally-scoped attack including the WINS vulnerability scanning, Veritas Backup Agent vulnerability scanning, and the MySQL Worm.

1 Introduction

Networks are increasingly subjected to threats that affect the reliability of critical infrastructure. These include Distributed Denial of Service attacks, such as the SCO DDoS attacks [2], and scanning worms, such as CodeRed [33] and Blaster [3]. The impact of these threats is profound, causing disruptions of real world infrastructure [26] and costing individual institutions hundreds of thousands of dollars to clean up [11]. To address the concerns raised by these threats, researchers have proposed a variety of global early warning systems whose goal is to detect and characterize these threats.

Unfortunately, the properties of these threats make them particularly difficult to address. First and foremost, they are globally scoped, respecting no geographic or topological boundaries. For example, at its peak, the Nimda worm created 5 billion infection attempts per day, which included significant numbers from Korea, China, Germany, Taiwan, and the US [33]. In addition, they can be exceptionally virulent and can propagate to the entire population of susceptible hosts in a matter of minutes. This was the case during the Slammer worm, in which the majority of the vulnerable population (75K+ susceptible hosts) was infected in less than 30 minutes [21]. This virulence is extremely taxing on network resources and creates side effects that pose problems even for those that are outside of the vulnerable population, such as the routing instability associated with the Slammer worm [17]. To make matters worse, these threats have the potential to be zero-day threats, exploiting vulnerabilities for which no signature or patch is available. For example victims of the Witty worm were compromised via their firewall software the day after a vulnerability in that software was publicized [31].

In order to address these properties, threat detection and classification systems are needed to provide detailed forensic information on new threats in a timely manner. As many threats propagate by scanning the IPv4 address space, researchers have turned to monitoring many addresses at the same time in order to quickly detect these threats [33, 32]. By monitoring large numbers of addresses, these systems can notably increase the probability of quickly detecting a new threat as it attempts to infect other hosts on the Internet [22]. However, as threats become increasingly complex, interacting with the infected hosts to ellict the important threat features, such as exploit, rootkits, or behavior, may require increasingly complex host emulation. This, coupled with the possibility of zero-day threats that may provide little or no warning for creating these emulated behaviors, may leave wide addresses monitoring systems unable to identify the important threat characteristics. In contrast, honeypot systems provide detailed insight into new threats by monitoring behavior in a controlled environment [5, 34]. By deploying honeypot systems with monitoring software, one can automatically generate detailed forensic profiles of malicious behavior [16]. Unfortunately, this detailed analysis comes at the expense of scalability, and hence time to detection.

An interesting potential approach to this problem is to forward requests destined to darknets back to an automated bank of honeypots [15, 38, 30]. While this architecture provides the promise of quickly generating detailed forensic information, there are still serious problems that need to be addressed. In particular, there is still the very important question of what requests to forward to the honeypots. For example, a darknet consisting of an unused /8 address block (roughly 16 million routable IP addresses) observes almost 4 Mbits/sec of traffic, much of which is TCP connection requests. While seemingly a small amount of traffic, each of these connection requests may require their own virtual machine. This load can cause scalability issues for both servicing the large number of requests and storing or evaluating the large quantity of data produced [38].

In this paper, we investigate the problem of filtering darknet traffic in order to identify connections worthy of further investigation. In particular, we analyze data from a large, distributed system of darknet monitors. We characterize the traffic seen by these monitors to understand the scalability bounds of a hybrid monitoring system that consists of distributed darknet monitors and a centralized collection of honeypots (or honeyfarm). In addition, we use these characterizations to guide the design of an algorithm that is effective at reducing large traffic rates into a small number of manageable events for the honeyfarm to process.

The main contributions of this work are:

- **Measurement and analysis of a large, distributed dark address monitor.** The measurements and characterizations presented in this paper are from a multi-year deployment of over 60 darknets in 30 organizations including academic institutions, corporations, and Internet service providers. This deployment represents a tremendous amount of diverse address space including over 17 million routeable addresses with blocks in over 20% of all routed /8 networks.

- **Identification of several key threat characteristics that bound the scalability of a hybrid system.** The scalability of a hybrid system depends on limiting the number of connections that need to be sent to the honeyfarm for analysis. By examining the behavior of threats at a large number of darknets we note two important characteristics:

  – A small fraction of the total source IPs observed at a single darknet are responsible for the overwhelming majority of the packets.

  – Most behavior consists of sources, and to some extent target services, that are not observable across darknets.

  From these characterizations we show that source-based filtering is an effective method of reduction for individual darknets, but fails to provide additional benefits when multiple darknets are combined together.

- **Creation and deployment evaluation of an effective algorithm for scaling hybrid architectures.** We create an algorithm that is both very effective in reducing the large amount of traffic seen by darknets to a small handful of events and is easily within the capabilities of the most modest honeyfarms. A broad production deployment of this algorithm over a three month period in 2005 provided analysis of five major global events, including the MySQL Worm and the scanning associated with the WINS vulnerability, as well as the Veritas Backup vulnerabilities.

The remainder of this paper is structured as follows: We begin by reviewing the related work in section 2. In section 3 we introduce our hybrid architecture and some of the challenges in building any similar system. We then examine the behavior of threats at individual dark address blocks in section 4. We observe these threats across darknets in section 5, and based on the insights from these measurements, we construct a filtering algorithm that we describe in section 6. In section 7 we show how this algorithm is effective at reducing large traffic rates to a small handful of events through a broad production deployment. We then finish with our conclusions in section 8.

2 Related Work

Historic approaches to the detection and characterization of network-based security threats fall into two categories; monitoring production networks with live hosts and monitoring unused address space (or darknets). In monitoring used networks, systems may choose to watch traffic directly [20] or watch abstractions of the data, such as flow records [13]. Security devices also provide an important source of these abstractions, and alerts from host-based anti-virus software [6], intrusion detection systems [10], and firewalls have been used as an effective means of addressing certain security threats. In contrast to monitoring live hosts, darknet monitoring consists of sensors that monitor blocks of unused address space. Because there are no legitimate hosts in a darknet, any observed traffic destined to such darknet must be the result of misconfiguration, backscatter from spoofed source addresses, or scanning from worms and other network probing. Methods for watching individual addresses with sacrificial hosts are often called honeypots [5, 34]. Techniques for monitoring much wider address blocks have a variety of names including network telescopes [22], blackholes [33], and darknets [8]. It should be noted that a limitation of this approach is that it relies on observing the target selection behavior of threats. As a result, threats that do not scan for new hosts, or threats that are specifically tailored to avoid unused blocks are not observed. Nevertheless, these techniques have been used with great success in observing denial of service activity [23], worms and scanning [32], as well as other malicious behavior.

In isolation, these techniques fail to completely address the scalability and behavioral fidelity requirements needed to monitor these threats. The scope of existing host-based techniques, such as host-based honeypots, anti-virus software, and host-based intrusion detection, is too small to capture global information such as the size of the infected population, or provide warning early in the growth phases. On the other hand, globally-scoped network sensors, such as network telescopes, do not interact sufficiently with the worm. As such, they lack enough information to characterize the vulnerability exploited and its effect on local machines. To be effective at assuring the availability of Internet resources, it is necessary to combine information from disparate network resources, each with differing levels of abstraction, into one unified view of a threat.

Acknowledging this need for both host and network views, two new approaches of combining these resources have evolved, aggregating fine-grained sensor measurements from a large numbers of sensors, and hybrid systems that use darknet monitors to concentrate connections to a centralized honeyfarm. Projects that aggregate data fall into two categories, those based on aggregating firewall logs or Intrusion Detection System (IDS) alerts across multiple enterprises [37, 36, 41], and those based on constructing and aggregating data from large numbers of honeypots [35, 25]. Hybrid systems [15, 38, 30] vary in how they perform the physical connection funneling, where and in what way they choose to filter the data, and in the diversity and amount of address space monitored. Of particular relevance is the recent work on the Potemkin Virtual Honeyfarm [39] in which the authors discuss a hybrid architecture with emphasis on a novel set of techniques for creating scalable per connection virtual machines. Their scalability gains are achieved by multiplexing across idleness in the network and by exploiting redundancies in the per-host state of the virtual machines. The gains reported vary widely based on work load (from a few hundred to a million destination IPs per physical host) but under realistic workloads a single physical host can support 50k-100k destinations IPs. This work is complimentary to ours in that we focus on limiting the number of connections seen by the honeyfarm while the Potemkin authors focus primarily on servicing these connections as efficiently as possible.

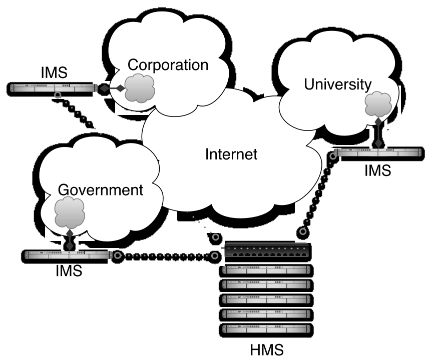


Figure 1: A Hybrid architecture with the distributed Internet Motion Sensor (IMS) with the centralized Host Motion Sensor (HMS).

3 A Hybrid Architecture for Monitoring Internet Security Threats

To provide both behavioral fidelity and global, broad coverage, we have proposed a hybrid architecture that is highly scalable but still delivers very accurate detection. This multi-resolution, hybrid architecture (shown in Figure 1) consists of two components: a collection of distributed darknet sensors (the Internet Motion Sensor or IMS), and a collection of host sensors (the Host Motion Sensor or HMS). In this architecture the IMS is used to monitor a broad, diverse set of darknets. When new activity is detected by the IMS, the connection is proxied back to the HMS for further in-depth analysis. The connection is relayed to virtual machine images running the application appropriate to the connection request. Thus, new and important threats are handed off and actually executed so the resulting activity can be monitored for new worm behavior.

By watching darknets, the traffic seen by the Internet Motion Sensor is pre-filtered to eliminate both the false positives in identifying malicious traffic and the scaling issues of other monitoring approaches. To analyze this traffic, the IMS sensors have both active and passive components. The active component responds to TCP SYN packets with a SYN-ACK packet to elicit the first data payload on all TCP streams. When a packet is received by a darknet sensor, the passive component computes the hash of the payload. If the hash doesn't match any previously observed signatures, then the payload is stored and the signature is added to the signature database. The Internet Motion Sensor architecture was introduced in 2005 [2], and it has been used to track new threats [3, 1] and to evaluate the importance of distributed darknet monitoring [7].

The Host Motion Sensor (HMS) is designed to provide additional forensic analysis in response to the changing threat landscape. It consists of three components: a virtual machine management module, a network detection module, and a host resource module. The virtual machine management module runs the target operating systems on top of a virtual machine. This module determines when to start a clean OS image, when to save an infected host image to disk, and it manages the working set of applications and operating systems. The network module, like the outbound connection profiling HoneyStat system [9], is responsible for looking for infected host behavior. It profiles the originating traffic from the honeypot and alerts on any outbound connection attempt not part of its normal behavior. Publicly available intrusion detection software [28] that matches traffic against known malicious signatures is used as an oracle for classifying any newly detected threats as "known" or "unknown". The final module is the host resource profiler. This system uses BackTracker [16] to build file and process dependency trees to both detect violations of policy and provide detailed forensic information on the files and processes created by an possible infection. Again, existing techniques are used to help identify new activities, in this case host anti-virus software [6].

There are several research problems associated with this or any hybrid approach that must be solved in order to ensure successful operation. These include:

- **Filtering Interactions.** Large darknets see staggering amounts of traffic that can not simply be redirected to a honeyfarm. In order to achieve scale, the amount of data presented to the honeyfarm must be significantly reduced.

- **Emulating Real Host Behavior.** As threats become increasingly sophisticated, detection systems must become correspondingly complex to elicit the proper response and to avoid fingerprinting. Understanding this arms race and the associated tradeoffs is key to any successful deployment.

- **Automating Forensics.** At the speed new threats are capable of spreading, the operational impact of human-scaled forensic analysis is minimal. Automated techniques are needed for generating actionable forensic information about a threat, including behavioral signatures describing activity at the network and/or host levels.

- **Managing Virtual Machines.** Even with advanced filtering mechanisms, the virtual machines are expected to handle a large number of requests and must be managed efficiently (as is discussed in the Potemkin Virtual Honeyfarm [39]). In addition, the effectiveness of the entire system is dependent on its ability to accurately represent the vulnerable population of interest.

In this section we discussed both the IMS and HMS components of our hybrid monitoring system as well as several of the research problems associated with any such hybrid system. In the next section, we focus on one of these research problems, reducing and filtering the interactions between the darknets and the honeyfarm.

4 Hybrid Scalability at Individual Darknets

For hybrid systems to be effective, they must make intelligent decisions about what darknet traffic to send to the honeyfarm. An idealized mechanism achieves scale by reducing redundant information and only forwarding one instance of each unique threat to the honeyfarm. Unfortunately, the mechanisms for determining what to handoff must make these decisions with the imperfect information available at a darknet monitor. In order to minimize overhead, a darknet monitor typically collects packets passively. As such, it has available elements from the network packet including the standard 5-tuple (source IP addresses, source port, destination IP address, destination port, and protocol), as well as any payload data seen for UDP, ICMP, or TCP backscatter packets. As mentioned previously, the IMS darknet sensor also collects the first payload packet for TCP connections through a scalable responder [2]. While other methods have been proposed for eliciting additional application information (for example, by building application responders via Honeyd [25]), in this paper we fix the data available for determining what to handoff and leave the issue of exploring more sophisticated information sources for future work.

In the following section, we explore the characteristics of these six elements (the five tuple and initial payload data) in order to determine how they affect the scalability of a hybrid system at individual darknets. We begin by examining the properties of individual darknets and in particular the behavior of source IP addresses. We provide these characterizations by looking at data from 14 darknet monitors ranging in size from a /25 monitor to a /17 monitor over a period of 10 days between August 18, 2004 and August 28, 2004. We then use these characterization to examine the effectiveness of proposed filtering techniques in reducing the connection which need to be evaluated by the honeyfarm.

4.1 Characterizing Individual Blocks

We begin by evaluating the source IP addresses seen at each darknet as a mechanism for determining bounds on the number of unique connections to be evaluated. As with

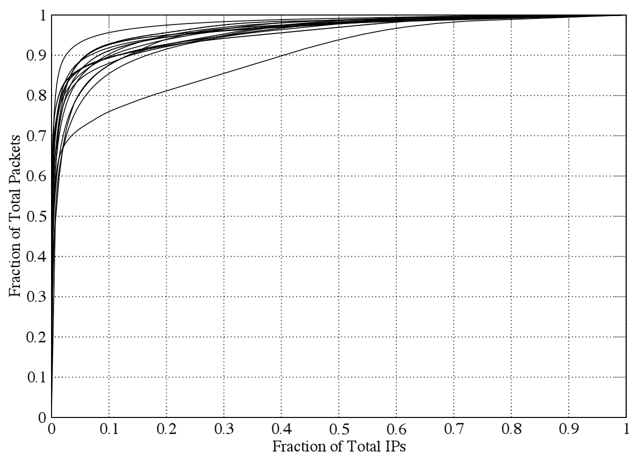


Figure 2: The contribution of individual IP to the total number of packets as seen at 14 darknets. Over 90% of the packets are from 10% of the source IP addresses.

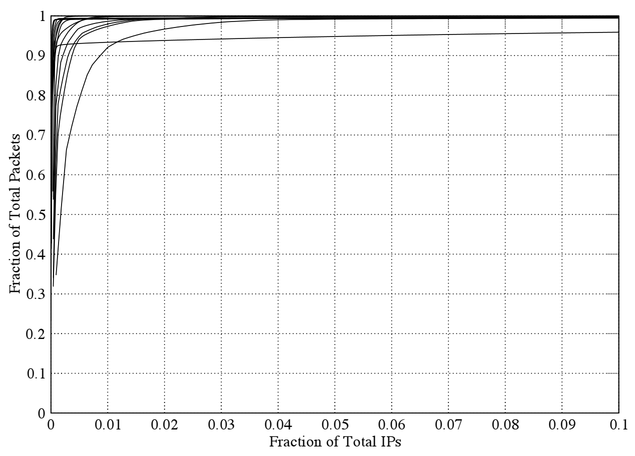


Figure 3: The contribution of a port to the total number of packets as seen at 14 darknets. Over 90% of the packets target .5% of the destination ports.

all the imperfect methods available at a darknet, source IP addresses have limitations in their ability to represent the number of unique attack sources. First, IP addresses do not represent individuals, as multiple users may use the same computer. Second, the mapping from IP address to computer is not static, so a computer may be represented multiple times with different IP addresses [32]. As in other studies [32, 24], we attempt to minimize these effects by performing analysis across small time frames of less than a day, or more often, less than an hour. However, the actual impact of dynamic addresses is not studied here.

The number of source IP addresses seen at each individual darknet varied greatly, with an inter-block mean of 75,530 sources and a variance of 92,843 sources over the 10 days. The minimum number observed in a single day was 1,345 sources with a maximum of 352,254 sources. Some of the wide variability in sources seen can be attributed to the effect of monitored darknet size. In our sample we had a mean block size of 5,385 IPs (roughly a /22), with the smallest darknet being a /25 and the largest a /17. However, even when normalizing to the smallest block size of /25, we have an inter-block mean of 40,540 sources and a variance of 30,381.

To understand how these source IP addresses behave, we examined the number of packets sent by each source IP address over the 10-day observation period at each of the 14 darknets. Figure 2 shows a surprising distribution: over 90% of the total packets observed at each darknet were sent from less than 10% of the source IP addresses seen at that darknet.

The other property that limits the number of unique connections to be evealuated is the number and types of services contacted. To explore this space, we examined the unique destination ports contacted over our 10-day observation period. As with sources, ports are imperfect measures. In particular, destination port analysis suffers from the inability to differentiate activities to the same port and to represent combinations of port activities (as is the case in multi-vector attacks) into a single action. Nevertheless these, serve as a coarse-grained approximation sufficient for exploring the size of the destination service space.

In our analysis we again see a great deal of variability based on darknet size, with a mean number of contacted ports of 17,780 and a variance of 20,397. The minimum number of ports seen was 1,097 at a /25 and the maximum was 59,960 at a /17. With maximums approaching the total number of destination ports allowed, we conjecture that many of the ports observed are simply due to ports scans. Nevertheless, unless the scanning is uniform in some way (e.g., sequential) it would be difficult for the darknet monitors to treat these packets differently. To understand the role these diverse destination ports play in darknet traffic, we investigated the distribution of these destination ports and their effect on the number of packets. Again we see a very striking result: over 90% of the packets are from .5% of the destination ports.

Despite the focused distributions, the cross product of the total unique source IP addresses and total destination ports is actually quite large. In order for us to efficiently scale, the source IP addresses must repeat similar actions. We therefore look at the behavior of the top 10% source IP addresses in terms of the number of destination ports they contact as well as the number of unique payloads they send. Figure 4 shows the number of ports contacted by 10% of the IP addresses at each of 14 darknets over a period of 10 days. At each darknet, over 55% of these source IP addresses contacted a single destination port, 90% contacted less than six, and 99% of the source IP addresses contacted less than 10. A very small fraction of these very active source IP addresses contacted considerably more ports. As

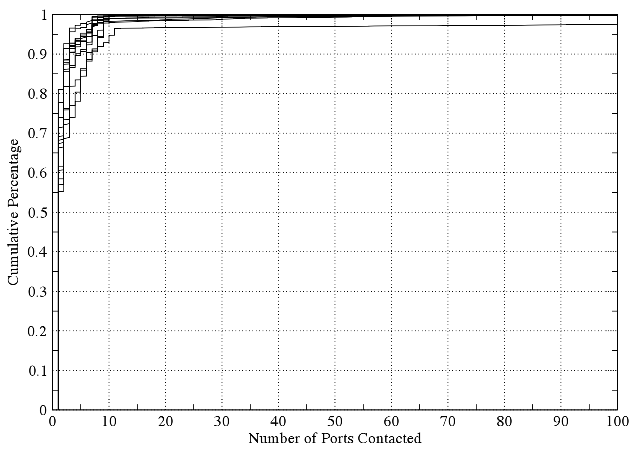


Figure 4: For the top 10 percent of IPs seen at each of the 14 darknets, the cumulative distribution function of the number of ports contacted.

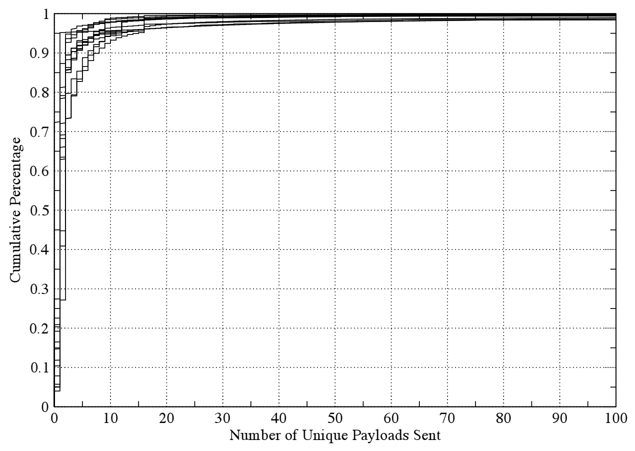


Figure 5: For the top 10 percent of IPs seen at each of the 14 darknets, the cumulative distribution function of the number of unique payloads sent.

expected, the fanout in the payloads sent is slightly larger, with 30% sending only one payload, 70% sending two or less, and 10 or less payloads seen by only 92%. In this analysis only the first payload of an action is considered. While a better differentiator of threats than ports, it may still under-represent the total number of events, as many services (e.g., Windows RPC) require multiple identical initiation packets. Nevertheless, both methods show that a significant fraction of the behaviors are the same for a source IP address, with the vast majority of the attacks involving multiple destination ports (multi-vector) and consisting of less than 10 contacted destination ports.

In this section we explored the source IP address behavior as seen by 14 individual darknets. We showed that a small fraction of the total source IP addresses are responsible for the vast majority of packets and that these sources

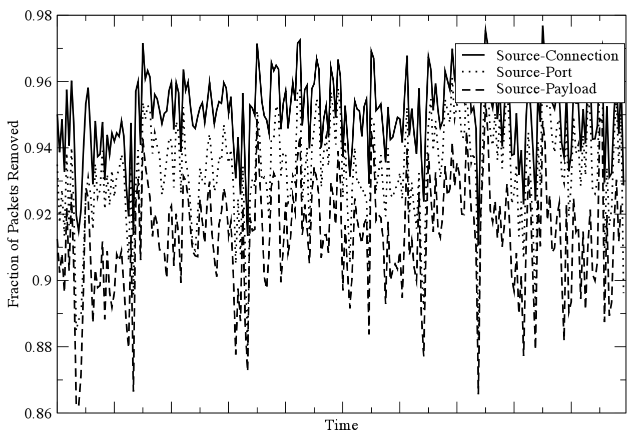


Figure 6: The reduction of Source-Connection, Source-Port, and Source-Payload filtering. Average effectiveness per hour for 14 darknets over a 10-day period with N=1.

are contacting the same, small handful of destination ports repeatedly. In the next section, we examine the effect of these results on several source-based filtering techniques and evaluate the practical value of applying these techniques to reduce packets seen at individual dark address blocks into a smaller number of unique events.

4.2 Source-Based Filtering

Recently a small body of work has emerged on filtering of the darknet traffic as a means to scale to large address blocks. This work has been published in the context of the iSink [38] project as well as the Internet Motion Sensor [2]. In the iSink work [38] the authors discuss two forms of filtering, random subnet selection and a sample and hold method. In subsequent work [24], the authors introduce four types of source-based filtering: source-connection, source-port, source-payload, and source-destination. The tradeoffs are discussed for each, including the effect of multi-stage (multiple connections to the same port) and multi-vector (multiple connections to different ports) based attacks on their accuracy. However, only source-connection is evaluated, and only at two darknets for a two-hour trace. The IMS authors have also discussed [2] preliminary results in using the contents of the first payload packets to reduce disk utilization and differentiate traffic.

The effect of three of the source-based methods is shown over a 10-day period at 14 darknets in Figure 6. In source-connection filtering, N connections from a single source are recorded, and all subsequent traffic from that source is ignored. Source-connection filtering serves as a baseline for determining the maximum reduction in any of the source-based approaches, as each contains the source as a component and is therefore limited to the number of unique

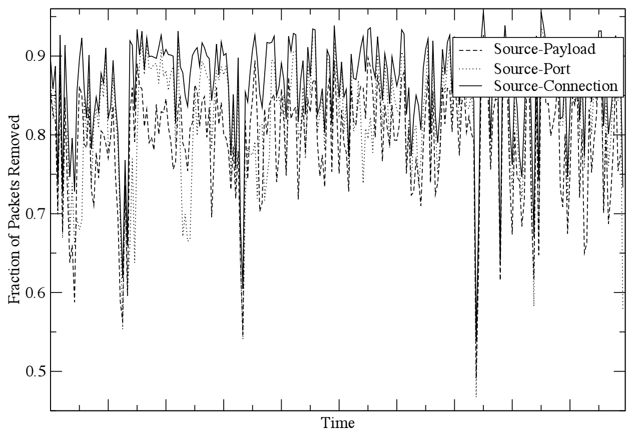


Figure 7: The minimum reduction of Source-Connection, Source-Port, and Source-Payload filtering. Minimum effectiveness per hour for 14 darknets over a 10-day period with *N*=1.

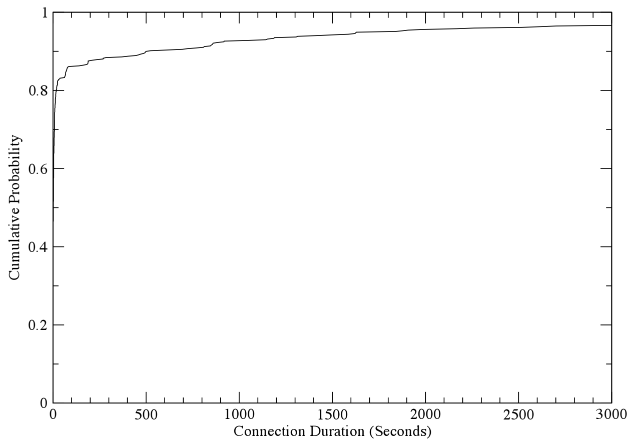


Figure 8: The cumulative distribution function of connection length from a Windows 2000 honeypot over a three-day period

sources in a period. In source-port filtering, *N* connections are maintained for every source and destination port pair. This method [24] eliminates the undercounting of events of source-connection filtering in the case of multi-vector activities, but multi-stage activities remain a problem, as connections to the same port may not be visible with a small number of *N* (e.g., *N*=1 will only record the first connection to a port). Finally, we have source-payload filtering in which the first *N* payloads sent by a source are used to define all future behavior by that source. We find that the least differentiating view of an event is seen with the source-connection filter. Because it counts any traffic from the same source as the same, it is undercounting the number of real events and therefore has the greatest reduction, a mean of 95%, across blocks. The more restrictive source-port filtering results in a mean reduction of 93% of the packets. Finally, the most differentiating view of events, source-payload, showed a mean reduction of 91%. On the whole, source-based techniques appear effective at reducing packets to a smaller number of events.

In comparing these results with the source-destination results reported previously [24] we see less effectiveness than the 99% reduction reported for values of $N = 1$, with even the least restrictive methods. While there are several possible explanations for this discrepancy, such as the difference in darknet block size (the blocks considered in [24] are /16s) we also report a great deal of variance, not only between darknets, but in a darknets over time. The intra-hour standard deviation for source-connection reduction was 2.7%, with source-port and source-payload at 3.1% and 3.9% respectively. Perhaps of more interest is the minimum value during each bin, as this is the run-time value that a hybrid filtering system would have to contend with. We show the minimum values in Figure 7. Here the minimum values drop to as low as 53.8% for source-connection filtering, and 46.7% and 49.1% for source-port and source-payload. In practice, the reductions observed may be less when applied to a runtime system that is making decisions about reduction and handoff based on the current observation period.

4.3 Effects on Hybrid Scalability

In order to understand the effect of source-based filtering on reducing the number of connections processed by a simple hybrid system, we considered the behavior at a single darknet. We applied source-payload filtering to the largest darknet over our 10-day observation window, a /17 darknet, and counted the unique source-payload events in one second bins. This analysis was designed to measure the total number of events per second a hybrid system consisting of a single darknet would be expected to handle. With averages less than 100, the system can expect to see bursts of several times that, with a single burst above 500 events per second being reported in the 10-day period.

Recall that the purpose of the hybrid system was to identify new threats and receive detailed analysis of those threats. As such, care must be taken to separate the cause of a specific event on the honeypots from its effect (for example, If a honeypot is processing several simultaneous connections, how can do we know which one successfully caused the infection?) In the worst case, we may need to separate every event and send it to a separate virtual host for processing. In this case, we now become bound not simply by the event per second rate but also by the duration of the event. To evaluate the cost of event duration, we examined the lengths of connections to an individual honeypot host, as shown in Figure 8. While roughly 77% of the connections were of zero length (a connection request and no more), the remaining distribution is very heavy tailed, with an average connection length of 400 seconds. In combination, these results indicate that a honeyfarm for a single /17 darknet block would need to handle from 40,000 to 200,000 simultaneous connections.

In the previous sections we explored the attack characteristics, as observed by individual darknets including the number of unique source addresses and destination ports. We examined the distribution of packets among these source IP addresses and destination ports and found that a supprisingly small number of source IP addresses and an even smaller number of destination ports dominated each darknet. We showed that these distributions make source-based filtering methods avery ppealing in reducing the traffic at individual blocks, but that there was a great deal of variance in the effectiveness of these methods over time. Nevertheless, we believe that these methods can be very helpful in reducing the number of packets at an individual darknet to a much smaller handful of connections.

5 Hybrid Scalability in Distributed Dark Address Blocks

For a hybrid system to be effective, our goals of detecting new threats and providing detailed analysis of these threats must be performed quickly when a new threat emerges. While we showed in the previous section that source-based filters could produce obtainable numbers of connections for handoff, the size of these darknets (e.g., /17) may be to small to provide quick detection of scanning worms. To achieve even further detection time reductions, a darknet monitoring system can choose to monitor a larger darknet, or combine multiple, distributed darknets. In practice, however, there is a limit on the size of a darknet (e.g., /8) and few of these large darknets exist. Moving beyond that size when such a large darknet is not available requires additional darknets to be aggregated together. This distributed darknet monitoring approach also has several additional benefits, including added resilience to fingerprinting, and insight into difference between darknets. In this section, we examine the properties of source IP addresses and destination ports across darknets to determine the effects of these properties on a hybrid system consisting of a distributed collection of darknet monitors.

5.1 Characterizing Distributed Darknets

We begin by looking at the number of source IP addresses at each of the 41 darknets during a 21-day period, from March 19th through April 9th, 2005. In Figure 9, we examine the cumulative unique source IP addresses seen per day. We see that blocks receive traffic from a few hundred (the /25 darknet) to a few million (the /8 darknet) unique source IP addresses per day. There is some overlap with previous

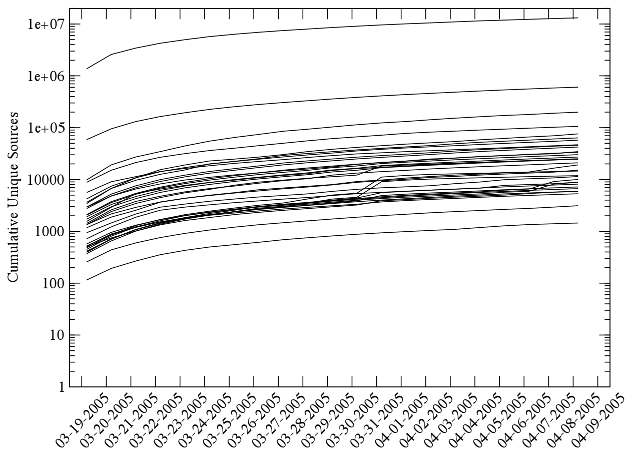


Figure 9: The number of cumulative unique sources per day, as viewed by 41 darknets from March 28th, 2005 to April 19th, 2005. Each line is a single darknet varying in size from a /25 to a /8

days, however, the forward difference still involves hundreds of thousands of hosts every day for the /8 darknet. This order of magnitude difference in the number of source IP addresses between the /8 and the /17 monitor discussed in the previous section adds a considerably larger number of events to evaluate for the larger darknet.

In order to see how the addition of darknets (each with their own number of unique source IP addresses over time) affects the aggregate number of sources in the hybrid monitor, we computed the overlap in unique source addresses between all darknets. Table 1 shows the average percentage of daily source IP address overlap (and standard deviation) in several of the medium to large darknets from the IMS system over a period of a month. A significant number of the source IP addresses seen at these darknets are not globally visible. The largest of the monitored darknets, the /8 darknet, consists mainly of source IP addresses not seen at any of the other darknets, as seen in the D/8 row. However, a much larger fraction of the source IP addresses at the other darknets do appear in the /8 darknet, as seen in the D/8 column. While this implies that many of the source IP address seen at a local darknet are captured by the /8, a significant faction of them are not; from 88% at one of the /22 darknets to 12% at one of the /18 darknets. In addition, the majority of the source IP addresses seen at the /8 are not seen anywhere else. This does not bode well for the scaling of a hybrid system, as the addition of each new darknet will be add a significant number of new source IP addresses and hence new connections to be evaluated.

Next we considered the space of the destination ports. For 31 darknets, we examined the top 10 destination ports, based on the number of packets, and compared these lists across darknets. Figure 10 shows the number of darknets that had a particular destination port in their top 10 list.

	A/18	B/16	C/16	D/23	D/8	E/22	E/23	F/17	G/17	H/17	H/18	H/22	I/20	I/21
A/18	100(0)	25(2)	58(5)	4(0)	78(5)	2(0)	4(0)	55(5)	28(2)	36(3)	28(3)	3(1)	16(2)	12(1)
B/16	23(3)	100(0)	38(5)	3(0)	54(8)	1(0)	3(0)	36(5)	20(3)	25(3)	18(2)	2(0)	10(1)	8(1)
C/16	23(2)	17(1)	100(0)	3(0)	78(6)	0(0)	2(0)	45(4)	20(2)	28(3)	20(1)	1(0)	9(0)	7(0)
D/23	10(0)	10(1)	20(1)	100(0)	30(1)	0(0)	1(0)	20(1)	10(0)	15(0)	11(0)	1(0)	5(0)	4(0)
D/8	2(0)	1(0)	5(0)	0(0)	100(0)	0(0)	0(0)	5(0)	1(0)	3(0)	2(0)	0(0)	0(0)	0(0)
E/22	10(2)	8(1)	13(1)	1(0)	12(1)	100(0)	3(0)	11(1)	9(1)	7(1)	6(0)	1(0)	7(0)	5(0)
E/23	25(4)	20(3)	33(5)	3(0)	34(5)	5(1)	100(0)	30(5)	21(3)	20(3)	16(2)	3(0)	16(2)	13(2)
F/17	23(1)	17(0)	48(1)	3(0)	82(1)	1(0)	2(0)	100(0)	22(0)	29(1)	21(0)	1(0)	9(0)	7(0)
G/18	20(1)	16(1)	36(1)	3(0)	51(2)	1(0)	2(0)	38(1)	100(0)	24(1)	16(1)	2(0)	9(0)	7(0)
H/17	16(0)	12(0)	31(1)	2(0)	53(1)	0(0)	1(0)	31(1)	14(0)	100(0)	27(2)	1(0)	8(0)	6(0)
H/18	20(1)	15(0)	37(2)	3(0)	56(2)	1(0)	2(0)	37(2)	16(0)	45(2)	100(0)	2(0)	11(0)	8(0)
H/22	7(2)	5(0)	9(1)	1(0)	16(3)	0(0)	1(0)	9(0)	6(0)	8(0)	6(1)	100(0)	3(0)	2(0)
I/20	11(1)	8(0)	16(1)	1(0)	16(1)	1(0)	1(0)	16(1)	8(0)	12(1)	10(0)	0(0)	100(0)	14(2)
I/21	13(1)	10(1)	20(1)	1(0)	19(1)	1(0)	2(0)	19(1)	11(1)	16(1)	12(1)	1(0)	21(4)	100(0)

Table 1: The average (and stddev) **percentage** overlap in source IP addresses between (row, column) medium to large darknets over a month period.

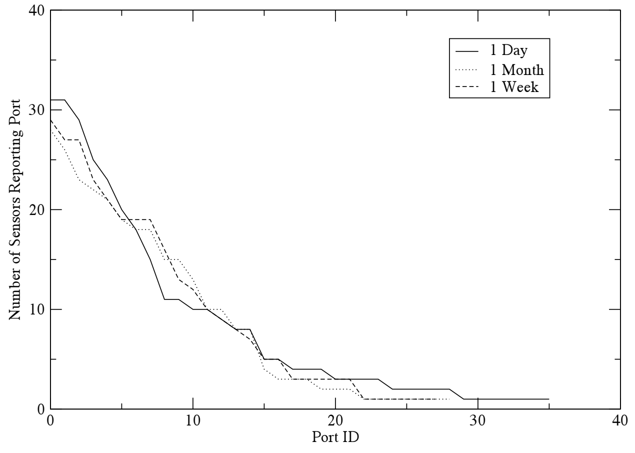


Figure 10: The number of darknets (of 31) reporting a port in the top 10 ports over a day, week, and month time frame.

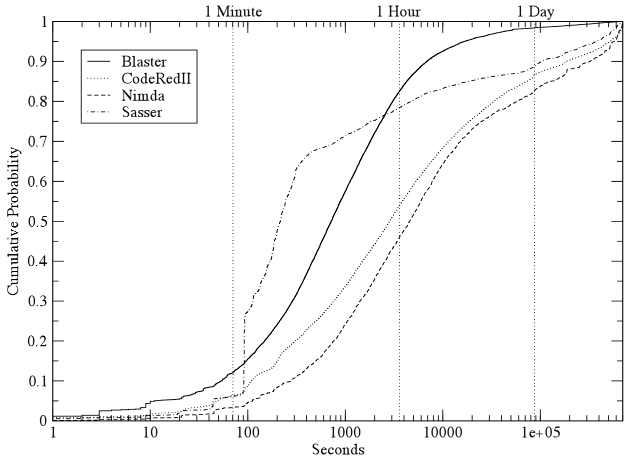


Figure 11: The duration of source IP address observations at the /8 darknet over a one week period for 4 known worms.

The analysis is performed for the top 10 destination ports over a day, top 10 destination ports over a week, and top 10 destination ports over a month. This figure shows that there are over 30 destination ports that appear on at least one darknet's top 10 list. A small handful of these destination ports appear across most darknets (1433, 445, 135, 139), but most of the destination ports appear at less then 10 of the darknets. As with the result seen with source IP addresses, the lack of consistency between darknets implies a broader number of connections to be evaluated, because of the broader number of non-overlapping destination services being contacted.

5.2 Understanding the Overlapping Size

The lack of overlap between various darknets in terms of source IP addresses, as well as destination ports, stems from four properties of the monitoring technique and the traffic.

The impact of *monitored block size, scanning rate*, and *observation time* on the probability of identifying a random scanning event. The effect of monitoring block size on the detection time of remote events is a well-studied phenomena [22]. Table 1 does show larger blocks with overlapping source IP addresses. However, the largest of these, although highly variable, only sees an average of 50% overlap.

The lifetime of the events. One possible explanation of this lack of overlap is that the events being observed are too short lived. To examine this, we looked at the behavior of four familiar worms whose (random and non-random) scanning behaviors are well known. Figure 11 shows this result. We recorded the observation lifetime of the source IP addresses across our /8 darknet. It should be noted that this method can only approximate the actual lifetime as the /8 darknet may only observe part of the worm's propagation behavior. With this caveat in mind, we note a significant faction of the source IP addresses seen at this darknet had a lifetime of less than a day. However, more than 50% had lifetimes of over an hour, which is sufficient at most scanning rates (greater than 10 connections per second) to be observed at the bigger darknets.

Targeted attacks. While the above explanations explain some of the lack of overlap, a significant number of source IPs are still not overlapping. It is our conjecture that these are representative of targeted behaviors and attacks. While this conjecture is difficult to validate without monitoring at the sources of these attacks, we can look at the distributions of destination ports for some insight as to whether the same services are being contacted across darknets. Recall that Figure 10 showed the number of darknets that report a port in their top 10 over increasing time frames of a day, week, and month. The results show that, although there are a few ports which are globally prevalent, a large number of the ports are seen over these time frames at very few blocks.

Environmental factors. In [7] we showed that a variety of factors, including filtering policy, propagation strategy, darknet visibility, and resource constraints, affect the mix of the global traffic a darknet should see. These properties provide subtle influences beyond the impact of the targeted behavior discussed above.

5.3 Effects on Hybrid Scalability

One of the drawbacks of any of the source-based techniques discussed in the previous section is their reliance on the source IP address as part of the filter. For a distributed technique based on these methods to be effective, source IP addresses must be prevalent across darknets. Unfortunately, our analysis shows this not to be the case. In order to quantify the effectiveness of source-based methods across darknets we consider the same 14 darknets as in the previous section. We choose source-port filtering and look at the reduction in unique (source, port) pairs across darknets over the same 10-day period. While this method was effective at reducing the packet space by 95%, there is only a modest 20% reduction when these methods are applied across darknets.

In this section we examined the properties of distributed darknets to understand the scaling properties of a hybrid system. We evaluated the properties of source IP addresses and destination ports across darknets and observed a significant number of non-overlapping destination ports and source IP addresses. While some of these differences are the result of well-known size and rate effects of darknet monitoring, other factors, such as the short on-time of random scanning hosts and the targeted nature of attacks observed at these darknets help to explain the additional differences. The impact of these non-intersecting events is that each individual new darknet added to a hybrid system is likely to significantly increase the total number of connections that need to be evaluated. If a hybrid system is to scale in this type of environment, a new definition of events and new methods of filtering are required.

6 Aggressive Distributed Filtering

In the previous sections, we discussed source-based filtering at individual darknets and the application of those filtering techniques to distributed darknets for the purpose of building a scalable hybrid system. We noted that source-based techniques were much less effective across darknets for two main reasons: source IP addresses are not typically visible across darknets in time frames that are useful and many attacks may be targeted at individual darknets only.

In this section we examine techniques for filtering that explicitly account for these observations. In particular, we examine the number of unique source IP addresses per destination port, and we examine the number of darknets reporting an event. We combine these variables with the technique of threshold-based alerting to provide a filter that passes traffic on global increases in the number of source IP addresses contacting a destination port.

Alerting of traffic changes by observing traffic to live hosts is a well studied area. Existing methods have alerted once traffic has reached a static threshold [27] or for thresholds that were adapted dynamically [14]. Further attempts to model traffic behavior that can be used for detecting changes include signal analysis [29], probabilistic approaches [18], and statistical approaches [12]. In this section we extend the existing techniques to look at adaptive threshold-based alerting for traffic to unused address space. Similar in vein to other source address distribution alerting mechanisms [40], we watch the distribution of unique source IP addresses to a specific port. Our work differs from this in several key ways: we modify the algorithm to explicitly allow for varying notions of current and historic behavior, we modify it to watch the distributions across multiple darknets, and we incorporate the notion of global versus local behavior.

Our current event identification scheme uses an adaptive threshold mechanism to detect new events. Every hour, each darknet is queried for the number of unique source IP addresses that have contacted it with destination port x (where x ranges over a list of destination ports we are monitoring). The event identification algorithm looks at the collected data and works as follows for each destination port x. For each hour, add up the number of unique source IP addresses contacting destination port x at each darknet. Scan over this data one hour at a time, comparing the average (per hour) over the event window (last event window hours) to the average over the history window (last event window $\times$ history factor) hours. If the ratio of event window average to history average is greater than the event threshold, an event is generated. These events are then filtered based on whether they are global or local, via the coverage threshold. The coverage threshold defines the number of darknets that would have generated an event individually for a threat. Events will not be generated more than once

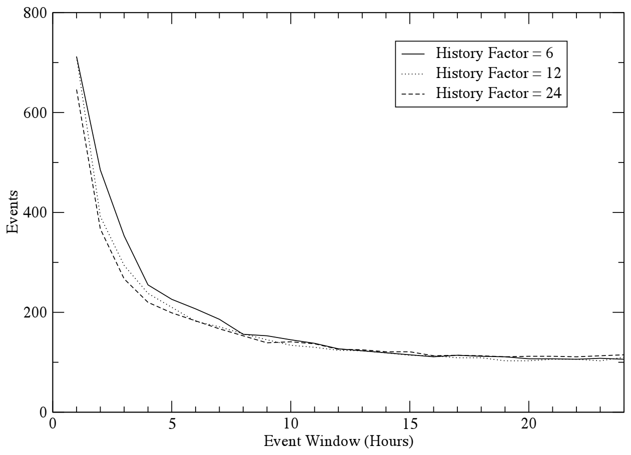


Figure 12: The effect of event window size on the number of events generated.

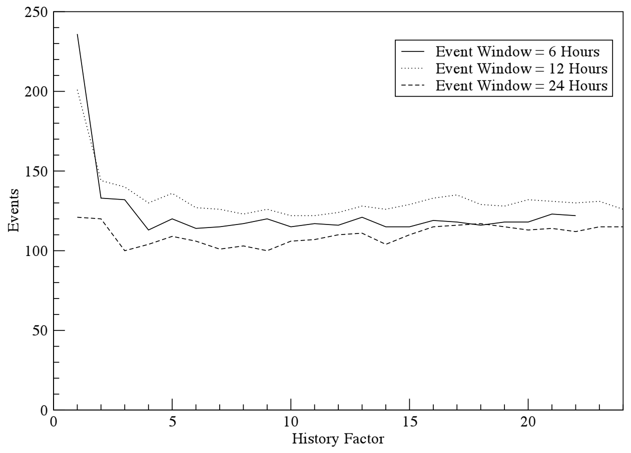


Figure 13: The effect of history factor on the number of events generated.

in a single event window. To protect against false positives on destination ports that have little or no traffic destined to them, whenever the sum of unique source IP addresses from all blocks to a specific destination port is less than one, the data point for that hour is set to 0.6. The event generation algorithm then is parameterized by four variables: the event window, the history window, the event threshold, and the coverage.

7 Evaluation and Deployment Results

In this section, we investigate the parameter space of the event identification algorithm and then evaluate it by comparing both the security events we identify and those identified by the community.

7.1 Parameterization

As discussed in the previous section, there are four basic parameters that impact the generation of a new event. In this section, we explore the tradeoffs associated with each of these parameters using data collected at 23 of the 60 IMS blocks from January 1st through April 30th, 2005.

The first parameter we explore is that of the event window. Recall that the event window defines the interval over which the current value is computed via a weighted moving average. Figure 12 shows the effect of the event window size in hours on the number of events generated for fixed window size and event threshold. The curve shows that the number of events vary from over 700 to a nerarly steady value of 100, with a significant reduction by a value of five hours. One possible explanation for this reduction is that many of the events are in fact short bursts, and longer event window sizes reduce the impact of single hour bursts.

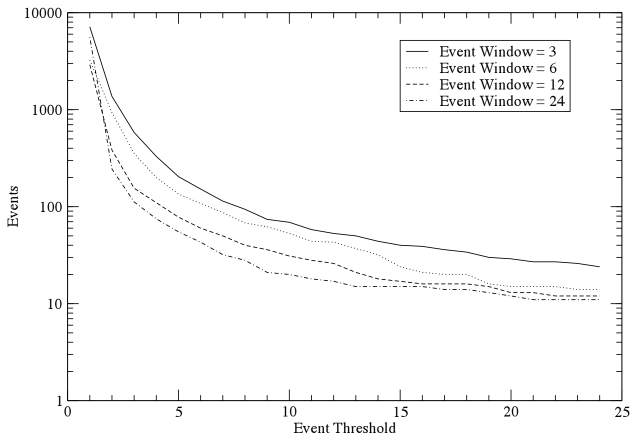


Figure 14: The effect of event threshold on the number of events generated.

History factor defines the period of time over which normal behavior is calculated via a weighted moving average. The effect of various history factor values on the number of events is explored in Figure 13. A history factor of two appears to reduce the number of events greatly. It is also interesting to note the crossover in event window size, with an event window of 12 hours creating *more* events than a smaller window of 6 hours. We find no significant difference in protocol or source distribution between these two sets. We are continuing to investigate two additional hypotheses: that sequential scanning activities need more time to reach additional darknets, and that there are frequency components of darknet traffic that are different than that of other Internet traffic.

Figure 14 shows the effect of the event threshold on the number of events. The event threshold indicates the degree to which the current value is different than the historic value. This parameter shows the largest impact on events

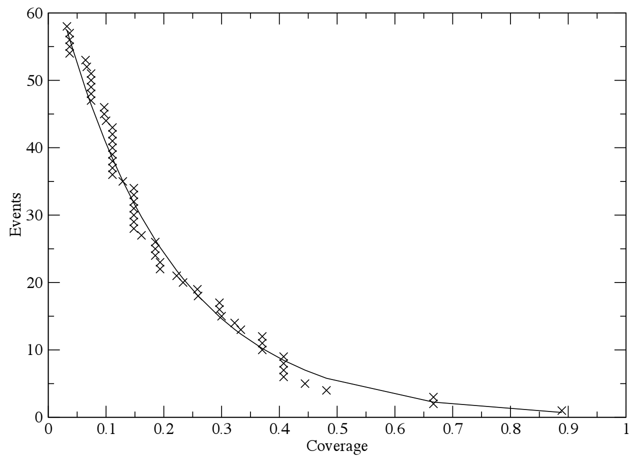


Figure 15: The effect of coverage on the number of alerts generated.

of any of the parameters, with a parameter of one generating an event for nearly every event window (event when the ratio of current to past is one). Drastic reductions in the number of events are seen by event thresholds of three to four.

Coverage represents the percentage of the total darknets monitored that would have individually reported an increase via our algorithm. Figure 15 shows the effect of various coverage values as a filter on the number of events. The majority of events generated by the algorithm are seen at a very small handful of darknets. The majority of reduction in the number events occurs when only considering events that are seen across more than 1/3 of the darknets.

7.2 Production Validation

To help validate our event generation algorithm, we compared the events identified by our system with those identified by the broader security community. It is important to highlight that our system has been in production use over the past few months and has identified several critical new threats. It is used by major network operators and governments around the world to quickly help identify new security events on the Internet.

Over an observation period of four months, the IMS system identified 13 unique events. Table 2 shows these events grouped by port. Included in these events are the TCP/42 WINS scanning, the TCP/6101 Veritas Backup Exec agent scanning, and the MySQL worm/bot on TCP/3306. These events demonstrate the quick onset, short duration, and global prevalence of many attacks. Figure 16 shows the normalized number of unique source addresses detected at the 23 IMS sensors for these events over time. As the operational test was underway before the completed parametization evaluation, the alerting algorithm used the following

Description	Port	Date	Multiple	Coverage
WINS	tcp42	01/13/05 17:31	5.36	0.4815
	tcp42	01/14/05 05:31	61.85	0.8889
	tcp42	01/14/05 17:31	9.77	0.6667
Squid and Alt-HTTP SYN Scan	tcp3128	02/05/05 11:31	7.73	0.4074
	tcp3128	02/05/05 23:31	18.19	0.4074
	tcp8080	02/05/05 10:51	7.53	0.4074
	tcp8080	02/05/05 22:51	20.95	0.3704
MYSQL	tcp3306	01/26/05 09:31	43.89	0.3704
	tcp3306	01/26/05 21:31	8.2	0.4444
	tcp3306	01/27/05 09:31	5.7	0.4074
Syn Scan	tcp5000	01/08/05 14:31	3.42	0.6667
Veritas	tcp6101	02/23/05 21:32	3.54	0.3704
	tcp6101	02/24/05 09:32	3.81	0.3333

Table 2: The interesting features identified by the algorithm since January of 2005. The “multiple” column specifies how many times larger the current window is compared to the history window. Coverage reports the percentage of sensors which would have alerted independently.

non-optimal parameters: alert window of 12 hours, history window of six days, and an alert threshold of three.

Beginning in December 2004, the IMS system observed a significant increase in activity on TCP port 42. This increased activity was notable because Microsoft had recently announced a new security vulnerability with the Windows Internet Name Service (WINS) server component of its Windows Server operating systems. The activity also followed a vulnerability report from Immunity Security on November 24, 2004 describing a remotely exploitable overflow in the WINS server component of Microsoft Windows. Payloads captured on TCP port 42 during the event include sequences that are byte-for-byte identical to exploit code found in the wild following the vulnerability announcement. Although the attack was very broadly scoped, it involved a small number of hosts and only lasted a day. Because the IMS was broadly deployed, it was able to quickly identify this small-scale event in time to enable additional monitoring capabilities.

Another quick attack occurred on January 11, 2005, when IMS observed a substantial increase in TCP port 6101 scanning. Approximately 600 sources were identified as aggressively probing many of the distributed IMS sensors. TCP port 6101 is used by the Veritas Backup Exec agent, a network service that allows for remote system backup. On December 16, 2004, iDefense announced a buffer overflow enabling remote system-level access. Then, on January 11, exploit code was published for the vulnerability and on the same day, IMS observed a large increase in activity on TCP/6101. Payloads captured from the attack include sequences that are byte-for-byte identical to the exploit code. Once again we see a very quick onset, suggesting the need for an automated system able to rapidly react to new threat activity.

Finally, in late January of 2005 IMS detected a worm/bot targeted at the MySQL database server [19]. The worm/bot propagated by randomly scanning for the Windows ver-

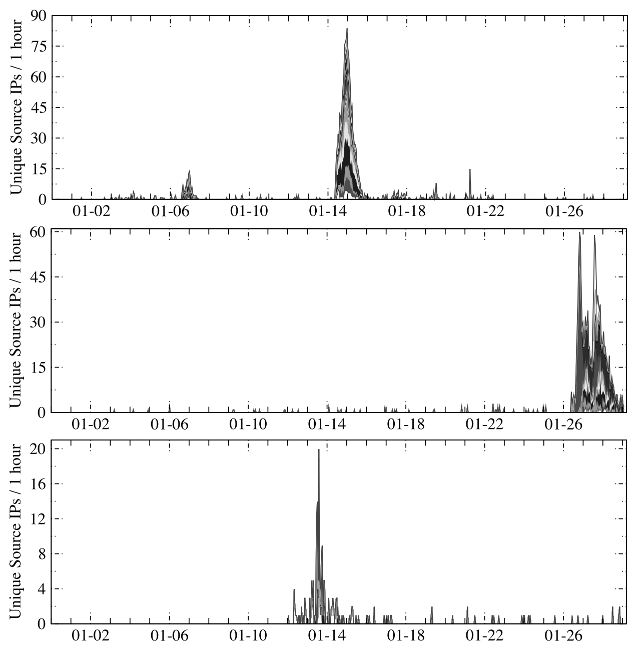


Figure 16: The unique source IP addresses over time across 23 darknets to various TCP destination ports. Widely publicized events are shown (from top to bottom TCP/42-WINS, TCP/3306-MYSQL, TCP/6101-VERITAS).

sion of MySQL on TCP port 3306. The threat was unique because it required interaction with a hidden IRC server before a new instance would start to propagate. The IRC channel was discovered by operators who shut it down, effectively eradicating the worm after a period of a few days. This event shows the importance of having more detailed information on a threat. Without observing the interaction of the worm/bot with the IRC server, there would be no way to discover the communication channel and thus find the simple way to stop the worm.

These three events help substantiate the detection approach and illustrate the value of automated distributed forensics. To further confirm these results and those listed in Table 2, we searched for correlations between the port, payloads, and time period of the events with reports from other security data sources. With the exception of the Altweb and Squid SYN scans, each of these events has been actively discussed in various security forums and corroborated by other monitoring projects [37, 4]. There is also evidence that the event identification algorithm did not miss any important events. Analysis of the NANOG, ISN, BUGTRAQ, FULL-DISCLOSURE, as well as the operator logs from ISC and CERT, do not show any major detectable events that were missed by the IMS system. In summary, there is strong evidence that the system was able to identify all the major events during the 4 month period. More time is clearly needed to fully validate the system, but the current data suggests the approach has excellent filtering qualities.

8 Conclusion

Global threats are changing at an amazing rate, requiring new innovative mechanisms for tracking and characterizing them. One approach to this problem involves hybrids of darknets with honeyfarms. There are numerous hurdles to this approach, but in this paper we investigated the most important of these, scaling interactions between the darknets and the honeyfarms. We examined several of the key characteristics of traffic at individual darknets that affect the scalability of a hybrid system. In particular, we showed that darknets are dominated by a small handful of source IP addresses contacting the same destination ports repeatedly. These characteristics allow source-based approaches to be quite effective at reducing the number of packets a block sees to a much smaller number of connections. We then showed, however, that this does not hold across darknets; neither the source IP addresses, nor to a lesser extent the destination ports, appear frequently across darknets. While some of this behavior can be explained by existing work on darknet size, scanning rate, and time to detection, we showed that much of it is the result of the targeted nature of the attacks observed as well as the short on-time of many random scanning hosts. This lack of overlap implies that every additional new block added to a distributed darknet monitor will likely bring with it its own sensor-specific events. For a large collection of darknets to be effective, a new view of events and new methods of filtering are required. We then constructed a filtering mechanism that takes these two factors into account by including distributions of source IP addresses, rather than the source IP addresses specifically, and the number of darknets observing an activity. We showed that this algorithm is effective by examining several recent events, such as activity associated with new WINS vulnerability, Veritas vulnerabilities, and the recent MY-SQL worm.

Acknowledgments

This work was supported by the Department of Homeland Security (DHS) under contract number NBCHC040146, the Advanced Research and Development Activity (ARDA) under contract number NBCHC030104, and by corporate gifts from Intel Corporation and Cisco Corporation. The authors would like to thank all of the IMS participants for their help and suggestions. We would also like to thank Joseph Rock, Jenn Burchill, Jose Nazario, Dug Song, Robert Stone, and G. Robert Malan of Arbor Networks and Larry Blunk, Bert Rossi, and Manish Karir at Merit Network for their assistance and support.

References

[1] Michael Bailey, Evan Cooke, Tim Battles, and Danny McPherson. Tracking global threats with the Internet Motion Sensor. 32nd Meeting of the North American Network Operators Group, October 2004.

[2] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[3] Michael Bailey, Evan Cooke, David Watson, Faranam Jahanian, and Jose Nazario. The Blaster Worm: Then and Now. *IEEE Security & Privacy*, 3(4):26–31, 2005.

[4] Lawrence Baldwin. My Net Watchman - Network Intrusion Detection and Reporting. `http://www.mynetwatchman.com/`, 2005.

[5] Bill Cheswick. An evening with Berferd in which a cracker is lured, endured, and studied. In *Proceedings of the Winter 1992 USENIX Conference: January 20 — January 24, 1992, San Francisco, California*, pages 163–174, Berkeley, CA, USA, Winter 1992.

[6] Fred Cohen. *A Short Course on Computer Viruses*. John Wiley & Sons, 2nd edition, April 1994.

[7] Evan Cooke, Michael Bailey, Z. Morley Mao, David Watson, and Farnam Jahanian. Toward understanding distributed blackhole placement. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04)*, New York, October 29 2004. ACM Press.

[8] Team CYMRU. The darknet project. `http://www.cymru.com/Darknet/index.html`, June 2004.

[9] David Dagon, Xinzhou Qin, Guofei Gu, Julian Grizzard, John Levine, Wenke Lee, and Henry Owen. Honeystat: Local worm detection using honeypots. In *Recent Advances in Intrusion Detection, 7th International Symposium, (RAID 2004)*, Lecture Notes in Computer Science, Sophia-Antipolis, French Riviera, France, October 2004. Springer.

[10] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[11] Andrea L. Foster. Colleges Brace for the Next Worm. *The Chronicle of Higher Education*, 50(28), 2004.

[12] Joseph L. Hellerstein, Fan Zhang, and Shahabuddin Shahabuddin. A statistical approach to predictive detection. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(1):77–95, January 2001.

[13] Cisco Systems Inc. Netflow services and applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`, 2002.

[14] Chuanyi Ji and Marina Thottan. Adaptive thresholding for proactive network problem detection. In *Third IEEE International Workshop on Systems Management*, pages 108–116, Newport, Rhode Island, April 1998.

[15] Xuxian Jiang and Dongyan Xu. Collapsar: A VM-based architecture for network attack detention center. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[16] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 223–236, Bolton Landing, NY, USA, October 2003. ACM.

[17] Lad, Zhao, Zhang, Massey, and Zhang. Analysis of BGP update surge during slammer worm attack. In *International Workshop on Distributed Computing: Mobile and Wireless Computing, LNCS*, volume 5, 2003.

[18] C. Leckie and R. Kotagiri. A probabilistic approach to detecting network scans. In *Proceedings of the Eighth IEEE Network Operations and Management Symposium (NOMS 2002)*, pages 359–372, Florence, Italy, April 2002.

[19] Robert Lemos. MySQL worm halted. `http://ecoustics-cnet.com.com/MySQL+worm+halted/2100-7349_3-5555242.html`, January 2005.

[20] G. Robert Malan and Farnam Jahanian. An extensible probe architecture for network protocol performance measurement. In *Proceedings of ACM SIGCOMM*, pages 177–185, Vancouver, British Columbia, September 1998.

[21] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[22] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Network telescopes. Technical Report CS2004-0795, UC San Diego, July 2004.

[23] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet denial-of-service activity. In *Proceedings of the Tenth USENIX Security Symposium*, pages 9–22, Washington, D.C., August 2001.

[24] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. Characteristics of Internet background radiation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 27–40. ACM Press, 2004.

[25] Niels Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, San Diego, CA, USA, August 2004.

[26] Reuters. Bank of America ATMs disrupted by virus. `http://archive.infoworld.com/togo/1.html`, January 2003.

[27] S. Robertson, E. Siegel, M. Miller, and Stolfo Stolfo. Surveillance detection in high bandwidth environments. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX 2003)*. IEEE Computer Society Press, April 2003.

[28] Martin Roesch. Snort — lightweight intrusion detection for networks. In USENIX, editor, *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII): November 7—12, 1999, Seattle, WA, USA*, Berkeley, CA, USA, 1999. USENIX.

[29] Amos Ron, David Plonka, Jeffery Kline, and Paul Barford. A signal analysis of network traffic anomalies. *Internet Measurement Workshop 2002*, August 09 2002.

[30] Stefan Savage, Geoffrey Voelker, George Varghese, Vern Paxson, and Nicholas Weaver. Center for Internet epidemiology and defenses. NSF CyberTrust Center Proposal, 2004.

[31] Colleen Shannon and David Moore. The spread of the Witty worm. *IEEE Security & Privacy*, 2(4):46–50, July/August 2004.

[32] Colleen Shannon, David Moore, and Jeffery Brown. Code-red: a case study on the spread and victims of an Internet worm. In *Proceedings of the Internet Measurement Workshop (IMW)*, December 2002.

[33] Dug Song, Rob Malan, and Robert Stone. A snapshot of global Internet worm activity. FIRST Conference on Computer Security Incident Handling and Response, June 2002.

[34] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

[35] Lance Spitzner et al. The honeynet project. `http://project.honeynet.org/`, June 2004.

[36] Symantec Corporation. DeepSight Analyzer. `http://analyzer.securityfocus.com/`, 2005.

[37] Johannes Ullrich. DSHIELD. `http://www.dshield.org`, 2000.

[38] Paul Barford Vinod Yegneswaran and Dave Plonka. On the design and use of Internet sinks for network abuse monitoring. In *Recent Advances in Intrusion Detection—Proceedings of the 7th International Symposium (RAID 2004)*, Sophia Antipolis, French Riviera, France, October 2004.

[39] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity and containment in the potemkin virtual honeyfarm. In *To appear in Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October 2005.

[40] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An effective architecture and algorithm for detecting worms with various scan techniques. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2004)*, San Diego, CA, February 2004. Internet Society.

[41] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the DOMINO overlay system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '04)*, San Diego, CA, February 2004.

Sparse Approximations for High Fidelity Compression of Network Traffic Data

William Aiello [†]
University of British Columbia
aiello@cs.ubc.ca

Anna Gilbert[§]
University of Michigan
annacg@umich.edu

Brian Rexroad
AT & T Labs
brexroad@att.com

Vyas Sekar [‡]
Carnegie Mellon University
vyass@cs.cmu.edu

Abstract

An important component of traffic analysis and network monitoring is the ability to correlate events across multiple data streams, from different sources and from different time periods. Storing such a large amount of data for visualizing traffic trends and for building prediction models of "normal" network traffic represents a great challenge because the data sets are enormous. In this paper we present the application and analysis of signal processing techniques for effective practical compression of network traffic data. We propose to use a *sparse approximation* of the network traffic data over a rich collection of natural building blocks, with several natural dictionaries drawn from the networking community's experience with traffic data. We observe that with such natural dictionaries, high fidelity compression of the original traffic data can be achieved such that even with a compression ratio of around 1:6, the compression error, in terms of the energy of the original signal lost, is less than 1%. We also observe that the sparse representations are stable over time, and that the stable components correspond to well-defined periodicities in network traffic.

1 Introduction

Traffic monitoring is not a simple task. Network operators have to deal with large volumes of data, and need to identify and respond to network incidents in real-time. The task is complicated even further by the fact that monitoring needs to be done on multiple dimensions and timescales. It is evident that network operators wish to observe traffic at finer granularities across different dimensions for a multitude of reasons that include: 1. real-time detection and response to network failures and isolating errant network segments, 2. real-time detection of network attacks such as DDoS and worms, and installation of filters to protect network entities, and 3. finer resolution root-cause analysis of the incidents and automated/semi-automated drill down of the incident.

To meet these requirements, we must be able to generate and store traffic data on multiple resolution scales in space (network prefixes and physical network entities such as links, routers), and in time (storing the traffic aggregates at multiple time resolutions). Such requirements naturally translate into increased operational costs due to the increased storage requirement. We often transport large portions of the historical data across a network to individual operators, import pieces of data into statistical analysis and visualization software for modeling purposes, and index and run queries against various historical databases for data drill down. Thus the management overhead involved in handling such large data sets, and the computational overhead in accessing and processing the large volumes of historical data also increases. We must reduce the storage size of the data, not only for efficient management of historical traffic data, but also to accommodate fine data resolution across space and time.

The compression techniques we investigate are "lossy" compression methods. For most network monitoring applications that utilize historical traffic data, it often suffices to capture salient features of the underlying traffic. We can thus afford some error by ignoring the low-energy stochastic components of the signal, and gain better compression using lossy compression techniques (as opposed to lossless compression methods such as gzip [11] which reduce the storage size of the data only and do not reduce the size of the input to monitoring applications). The overall goal of such compression techniques is to obtain high fidelity (i.e. low error) representations with as little storage as possible.

In particular, we use a compression method called sparse representation over redundant dictionaries. A visual inspection of aggregated network traffic for many high vol-

[†]Majority of this work was done when the author was a member of AT & T Labs-Research

[‡]Majority of this work was done when the author was a research intern at AT & T Labs-Research

[§]Majority of this work was done when the author was a member of AT & T Labs-Research. The author is supported by an Elizabeth Crosby Faculty Fellowship.

ume ports reveals three components. First, there is a natural diurnal variation for many ports and/or other periodic variations as well. Second, there are spikes, dips, and other components of the traffic that appear to be the result of non-periodic events or processes. Finally, the traffic appears to be stochastic over small time scales with variance much smaller than the periodic variations for high volume ports. Representing a signal with all three components using a single orthonormal basis, such as a Fourier basis or a wavelet representation is not likely to yield good compression: a basis that represents periodic signals well will not represent non-periodic signals efficiently and vice versa. The methods presented in this paper allow us to use two or more orthonormal bases *simultaneously*. A set of two or more orthonormal bases is called a redundant dictionary. Hence, with an appropriate set of orthonormal bases as the redundant dictionary, the periodic and the significant non-periodic portions of the traffic time series can both be represented efficiently within the same framework.

Sparse representation or approximation over redundant dictionaries does not make assumptions about the underlying distributions in the traffic time series. As a result, sparse approximation can guarantee high fidelity regardless of changes in the underlying distributions. In addition, there are highly efficient, provably correct algorithms for solving sparse approximation problems. These algorithms scale with the data and can be easily adapted to multiple sources of data. They are greedy algorithms, known as matching or orthogonal matching pursuit.

The primary contribution of this paper is a rigorous investigation of the method of sparse representation over redundant dictionaries for the compression of network time series data. We propose and evaluate several redundant dictionaries that are naturally suited for traffic time series data. We conclude that these methods achieve significant compression with very high fidelity across a wide spectrum of traffic data. In addition, we also observe that the sparse representations are stable, not only in terms of their selection in the sparse representation over time but also in terms of the individual amplitudes in the representation. These stable components correspond to well-defined periodicities in network traffic, and capture the natural structure of traffic time series data. To the best of our knowledge, this is the first thorough application of sparse representations for compressing network traffic data.

We discuss related work in Section 2, and present a overall motivation for compression in Section 3. In Section 4 we describe in more detail the framework of matching (greedy) pursuit over redundant dictionaries. Section 5 describes our traffic data set, derived from a large Internet provider. We evaluate the efficacy of our compression techniques in Section 6. Section 7 presents some network traffic monitoring applications that demonstrate the utility of the compression methods we used. Section 8 discusses the scope for improving the compression, before we conclude in Section 9.

2 Related Work

Statisticians concern themselves with subset selection in regression [13] and electrical engineers use sparse representations for the compression and analysis of audio, image, and video signals (see [4, 6, 12] for several example references).

Lakhina, et al. [9, 10] examine the structure of network traffic using Principal Component Analysis (PCA). The observations in our work provide similar insight into the structure of network traffic. There are two compelling reasons for using sparse approximations over redundant dictionaries, as opposed to PCA alone, for obtaining similar fidelity-compression tradeoffs. First, the description length for sparse approximation is much shorter than for PCA, since the principal vectors require substantially more space to represent than simple indices into a dictionary. Second, PCA like techniques may capture and identify the (predominant) structure across all measurements, but may not be adequate for representing subtle characteristics on individual traffic aggregates.

Barford, et al. [1] use pseudo-spline wavelets as the basis wavelet to analyze the time localized normalized variance of the high frequency component to identify signal anomalies. The primary difference is our application of signal processing techniques for compressing network traffic data, as opposed to using signal decomposition techniques for isolating anomalies in time series data.

There are several methods for data reduction for generating compact traffic summaries for specific real-time applications. Sketch based methods [8] have been used for anomaly detection on traffic data, while Estan et al. [3] discuss methods for performing multi-dimensional analysis of network traffic data. While such approaches are appealing for real-time traffic analysis with low CPU and memory requirements, they do not address the problems of dealing with large volumes of historical data that arise in network operations. A third, important method of reducing data is sampling [2] the raw data before storing historical information. However, in order for the sampled data to be an accurate reflection of the raw data, one must make assumptions regarding the underlying traffic distributions.

3 Compression

It is easy to (falsely) argue that compression techniques have considerably less relevance when the current cost of (secondary) storage is less than \$1 per GB. Large operational networks indeed have the unenviable task of managing many terabytes of measurement data on an ongoing basis, with multiple data streams coming from different routers, customer links, and measurement probes. While it may indeed be feasible to collect, store, and manage such a large volume of data for small periods of time (e.g. for the last few days), the real problem is in managing large volumes of historical data. Having access to historical data is a crucial part of a network operator's diagnostic toolkit. The historical datasets are typically used for building prediction models for anomaly detection, and also for building visual diagnostic aids for network operators. The storage requirement increases not only because of the need for access to large volumes of historical traffic data, but also the pressing need for storing such historical data across different spatial and temporal resolutions, as reference models for fine-grained online analysis.

It may be possible to specify compression and summarization methods for reducing the storage requirement for specific traffic monitoring applications that use historical data. However, there is a definite need for historical reference data to be stored at fine spatial and temporal resolutions for a wide variety of applications, and it is often difficult to ascertain the set of applications and diagnostic techniques that would use these datasets ahead of time. The compression techniques discussed in this paper have the desirable property that they operate in an application-agnostic setting, without making significant assumptions regarding the underlying traffic distributions. Since many traffic monitoring applications can tolerate a small amount of error in the stored values, lossy compression techniques that can guarantee a high fidelity representation with small storage overhead are ideally suited for our requirements. We find that our techniques provide very accurate compressed representations so that there is only a negligible loss of accuracy across a wide spectrum of traffic monitoring applications.

The basic idea behind the compression techniques used in this paper is to obtain a sparse representation of the given time series signal using different orthonormal and redundant bases. While a perfect lossless representation can be obtained by keeping all the coefficients of the representation (e.g. using all Fourier or wavelet coefficients), we can obtain a compressed (albeit lossy) representation by only storing the high energy coefficients, that capture a substantial part of the original time series signal.

Suppose we have a given time series signal of length N. For example, in our data set consisting of hourly aggregates of traffic volumes, N=168 over a week, for a single traffic metric of interest. We can obtain a lossless representation by using up a total storage of $N \times k$ bits, where k represents the cost of storing each data point. Alternatively, we can obtain a sparse representation using m coefficients using a total storage space of $m \times k' + |D|$ bits, where the term $|D|$ represents the length of the dictionary used for compression, and k' represents the cost of storing the amplitude associated with each coefficient. The $|D|$ term represents the cost of storing the list of selected indices as a bit-vector of length equal to the size of the dictionary. The length of the dictionary $|D|$ is equal to αN, with the value α being one for an orthonormal basis (e.g., Fourier, Wavelet, Spike) or equal to two in the case of a redundant dictionary consisting of Fourier and Spike waveforms. The effective compression ratio is thus $(mk' + \alpha N)/(Nk)$. Assuming $k \approx k'$ (the cost of storing the raw and compressed coefficients are similar) and $\alpha \ll k$ (the values in consideration are large integers or floats), the effective compression (even with this naive encoding) is approximately equal to m/N [1]. The primary focus of this paper is not to come up with an optimal encoding scheme for storing the m coefficients to extract the greatest per-bit compression. Rather we wish to explore the spectrum of signal compression techniques, using different natural waveforms as dictionaries for achieving a reasonable error-compression tradeoff.

A natural error metric for lossy compression techniques in signal processing is the energy of the residual, which is the vector difference between the original signal and the compressed representation. Let S be the original signal and C_s represent the compressed representation of S. The signal $R = S - C_s$ represents the residual signal. We use the following relative error metric $\frac{\|R\|^2}{\|S\|^2}$ where $\| \cdot \|$ represents the L_2 (Euclidean) norm of a vector. The error metric represents the fraction of the energy in the original signal that is not captured in the compressed model. For example, a relative error of 0.01 implies that the energy of the residual signal (not captured by the compressed representation) is only 1% of the energy of the original signal. Our results indicate that we can achieve high fidelity compression for more than 90% of all traffic aggregates, with a relative error of less than 0.01 using only $m = 30$ coefficients, for the hourly aggregates with $N = 168$. Since a m-coefficient representation of the signal implies a compression ratio of roughly m/N, with $N = 168$, a 30-coefficient representation corresponds to a compression ratio of roughly 1:6.

Consider the following scenario. An operator wishes to have access to finer resolution historical reference data collected on a per application port basis (refer Section 5 for a detailed description of the datasets used in this paper). Suppose the operator wants to improve the temporal granularity by going from hourly aggregates to 10 minute aggregates. The new storage requirement is a non-negligible $60/10 \times X = 6X$, where X represents the current storage requirement (roughly 1GB of raw data per router per week). Using the compression techniques presented in this paper, by finding small number of dictionary components to represent the time series data, the operator can easily offset this increased storage cost.

Further, we observe (refer Section 8.2) that moving to finer temporal granularities does not actually incur substantially higher storage cost. For example we find that the same fidelity of compression (at most 1% error) can be obtained for time-series data at fine time granularity (aggregated over five minute intervals) by using a similar number of coefficients as those used for data at coarser time granularities (hourly aggregates). Thus by using our compression techniques operators may in fact be able to substantially cut down storage costs, or alternatively use the storage "gained" for improving spatial granularities (collecting data from more routers, customers, prefixes, etc).

In the next section, we present a brief overview on the use of redundant dictionaries for compression, and present a greedy algorithm for finding a sparse representation over a redundant dictionary.

4 Sparse Representations over Redundant Dictionaries

One mathematically rigorous method of compression is that of sparse approximation. Sparse approximation problems arise in a host of scientific, mathematical, and engineering settings and find greatest practical application in image, audio, and video compression [4, 6, 12], to name a few. While each application calls for a slightly different problem formulation, the overall goal is to identify a good approximation involving a few elementary signals—a *sparse* approximation. Sparse approximation problems have two characteristics. First, the signal vector is approximated with a linear model of elementary signals (drawn from a fixed collection of several orthonormal bases). Second, there is a compromise between approximation error (usually measured with Euclidean norm) and the number of elementary signals in the linear combination.

One example of a redundant dictionary for signals of length N is the union

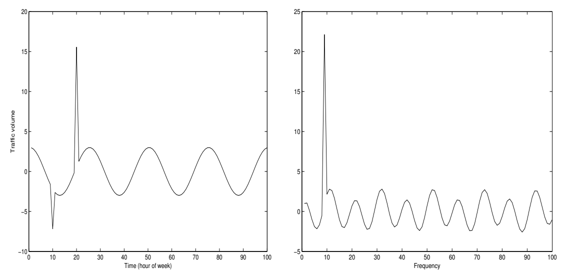
$$\mathcal{D} = \left\{ \cos\Big(\frac{\pi k(t+\frac{1}{2})}{N}\Big) \right\} \bigcup \left\{ \delta_k(t) \right\},$$

where $k = 0, \ldots, N-1$, of the cosines and the spikes on N points. The "spike" function $\delta_k(t)$ is zero if $t \neq k$ and is one if $t = k$. Either basis of vectors is complete enough to represent a time series of length N but it might take more vectors in one basis than the other to represent the signal. To be concrete, let us take the signal

$$X(t) = 3\cos\Big(\frac{\pi 8(t+\frac{1}{2})}{100}\Big) - 5\delta_{10}(t) + 15\delta_{20}(t)$$

plotted in Figure 1(a). The spectrum of the discrete cosine transform (DCT) of X is plotted in Figure 1(b). For this example, all the coefficients are nonzero. That is, if we write

$$X(t) = \frac{1}{100}\sum_{k=0}^{99} \hat{X}(k)\cos\Big(\frac{\pi k(t+\frac{1}{2})}{100}\Big)$$



(a) An example signal X which has a short representation over the redundant dictionary $\mathcal{D}$.

(b) The discrete cosine transform (DCT) of the example signal X.

Figure 1: The example signal X and its discrete cosine transform (DCT).

as a linear combination of vectors from the cosine basis, then all 100 of the coefficients $\hat{X}(k)$ are nonzero. Also, if we write $X(t)$ as a linear combination of spikes, then we must use almost all 100 coefficients as the signal $X(t)$ is nonzero in almost all 100 places. Contrast these two expansions for $X(t)$ with the expansion over the redundant dictionary $\mathcal{D}$

$$X(t) = 3\cos\Big(\frac{\pi 8(t+\frac{1}{2})}{100}\Big) - 5\delta_{10}(t) + 15\delta_{20}(t).$$

In this expansion there are only three nonzero coefficients, the coefficient 3 attached to the cosine term and the two coefficients associated with the two spikes present in the signal. Clearly, it is more efficient to store three coefficients than all 100. With three coefficients, we can reconstruct or decompress the signal exactly. For more complicated signals, we can keep a few coefficients only and obtain a good approximation to the signal with little storage. We obtain a high fidelity (albeit lossy) compressed version of the signal. Observe that because we used a dictionary which consists of simple, natural building blocks (cosines and spikes), we need not store 100 values to represent each vector in the dictionary. We do not have to write out each cosine or spike waveform explicitly.

Finding the optimal dictionary for a given application is a difficult problem and good approximations require domain specific heuristics. Our contribution is the identification of a set of dictionaries that are well-suited for compressing traffic time-series data, and in empirically justifying the choice of such dictionaries. Prior work on understanding the dimensionality of network traffic data using principal component analysis [10] identifies three types of eigenflows: periodic, spikes, and noise. With this intuition, we try different dictionaries drawn from three basic waveforms: periodic functions (or complex exponentials), spikes, and wavelets. Dictionaries that are comprised of these constituent signals are descriptive enough to capture the main types of behavior but not so large that the algorithms are unwieldy.

4.1 Greedy Pursuit Algorithms

A greedy pursuit algorithm at each iteration makes the best local improvement to the current approximation in hope of obtaining a good overall solution. The primary algorithm is referred to as Orthogonal Matching Pursuit (OMP), described in Algorithm 4.1. In each step of the algorithm, the current best waveform is chosen from the dictionary to approximate the residual signal. That waveform is then subtracted from the residual and added to the approximation. The algorithm then iterates on the residual. At the end of the pursuit stage, the approximation consists of a linear combination of a small number of basic waveforms. We fix some notation before describing the algorithm. The dictionary $\mathcal{D}$ consists of d vectors $\boldsymbol{\varphi}_j$ of length N each. We write these vectors $\boldsymbol{\varphi}_j$ as the rows in a matrix $\boldsymbol{\Phi}$ and refer to this matrix as the dictionary matrix. OMP is one of the fastest[2] provably correct algorithm for sparse representation over redundant dictionaries, assuming that the dictionary satisfies certain geometric constraints [5] (roughly, the vectors in the dictionary must be almost orthogonal to one another). The algorithm is provably correct in that if the input signal consists of a linear combination of exactly m vectors from the dictionary, the algorithm finds those m vectors exactly. In addition, if the signal is not an exact combination of m vectors but it does have an optimal approximation using m vectors, then the algorithm returns an m-term linear combination whose approximation error to the input signal is within a constant factor of the optimal approximation error. If we seek m vectors in our representation, the running time of OMP is $O(mdN)$. Dictionaries which are unions of orthonormal bases (which meet the geometric condition for the correctness of OMP), are of size $d = kN$, so the running time for OMP with such dictionaries is $O(mkN^2)$.

Algorithm 4.1 (OMP)
INPUT:

- *A $d \times N$ matrix $\boldsymbol{\Phi}$*
- *A vector $\boldsymbol{v}$ of measurements of length N*
- *The desired number of terms m in the compressed signal*

OUTPUT:

- *A set of m indices $\lambda_1, \ldots, \lambda_m$*
- *An N-dimensional residual $\boldsymbol{r}_m$*

PROCEDURE:

1. *Initialize the residual $\boldsymbol{r}_0 = \boldsymbol{v}$ and the iteration counter $t = 1$.*
2. *Find the index λ_t of the vector with the largest dot product with the current residual*
$$\lambda_t \;=\; \operatorname{argmax}_j \; |\langle \boldsymbol{r}_{t-1}, \boldsymbol{\varphi}_j \rangle| \,.$$
3. *Let $\boldsymbol{P}_t$ be the orthogonal projection onto the span of the current vectors* $\operatorname{span}\{\boldsymbol{\varphi}_\lambda : \lambda_1, \ldots, \lambda_t\}$. *Calculate the new residual:*
$$\boldsymbol{r}_t = \boldsymbol{v} - \boldsymbol{P}_t\,\boldsymbol{v}.$$
4. *Increment t, and return to Step 2 if $t < m$.*

Note that if we had a single orthonormal basis as the dictionary $\mathcal{D}$, the representation obtained using Algorithm 4.1 is exactly the same as the projection onto the orthonormal basis. For example, if we just had a Fourier basis, the coefficients obtained from a regular Fourier transform would exactly match the coefficients obtained from the matching pursuit procedure.

5 Data Description

The primary data set we have used for evaluating our methods consists of traffic aggregates collected over a 20 week period (between January and June 2004) at a large Tier-1 Internet provider's IP backbone network. The dataset consists of traffic aggregates in terms of flow, packet, and byte counts. The dimensions of interest over which the aggregates are collected are:

- TCP Ports: Traffic to and from each of the 65535 TCP ports.
- UDP Ports: Traffic to and from each of the 65535 UDP ports.
- Aggregated Network Prefixes: Traffic to and from network prefixes aggregated at a set of predefined network prefixes.

The traffic aggregates were generated from flow records using traffic collection tools similar to Netflow [14], aggregated over multiple links in the provider's Internet backbone. In this particular data set, the traffic volume counts are reported on an hourly basis. For example, for each TCP port the data set contains the total number of flows, packets, and bytes on that port. The data set aggregates each metric (i.e., flows, packets, and bytes) for both incoming (i.e., traffic with this port was the destination port) and outgoing traffic (i.e., traffic with this port as the source port). Such per-port and per-prefix aggregates are routinely collected at many large ISPs and large enterprises for various traffic engineering and traffic analysis applications.

It is useful to note that such data sets permit interesting traffic analysis including observing trends in the traffic data, and detecting and diagnosing anomalies in the network data. For many types of network incidents of interest (outages, DoS and DDoS attacks, worms, viruses, etc.) the dataset has sufficient spatial granularity to diagnose anomalies. For example, the number of incoming flows into specific ports can be an indication of malicious scanning activity or worm activity, while the number of incoming flows into specific prefixes may be indicative of flash-crowds or DoS attacks targeted at that prefix.

For the following discussions, we consider the data in week long chunks, partly because a week appears to be the smallest unit within which constituent components of the signal manifest themselves, and also because a week is a convenient time unit from an operational viewpoint.

6 Results

In this section, we demonstrate how we can use sparse approximations to compress traffic time series data. We look at the unidimensional aggregates along each port/protocol pair and prefix as an independent univariate signal. In the following sections, unless otherwise stated, we work with the total number of incoming flows into a particular port. We observe similar results with other traffic aggregates such as the number of packets and the number of incoming bytes incoming on each port, and for aggregated counts for the number of outgoing flows, packets, bytes on each port—we do not present these results for brevity. We present the results only for the TCP and UDP ports and note that the compression results for aggregated address prefixes were similar.

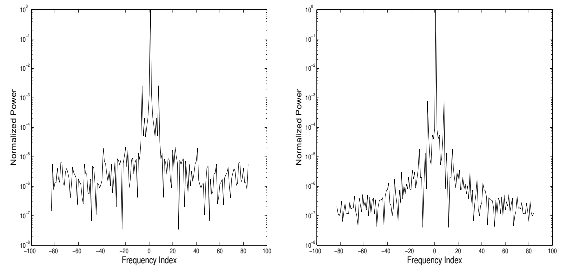
Since an exhaustive discussion of each individual port would be tedious, we identify 4 categories of ports, predominantly characterized based on the applications that use these ports. For each of the categories the following discussion presents results for a few canonical examples.

1. High volume, popular application ports (e.g., HTTP, SMTP, DNS).
2. P2P ports (e.g., Kazaa, Gnutella, E-Donkey).
3. Scan target ports (e.g., Port 135, Port 139) .
4. Random low volume ports.

6.1 Fourier Dictionary

Our first attempt at selecting a suitable dictionary for compression was to exploit the periodic structure of traffic time series data. A well known fact, confirmed by several measurements [9, 10, 15], is the fact that network traffic when viewed at sufficient levels of aggregation exhibits remarkably periodic properties, the strongest among them being the distinct diurnal component. It is of interest to identify these using frequency spectrum decomposition techniques (Fourier analysis). It is conceivable that the data can be compressed using a few fundamental frequencies, and the traffic is essentially a linear combination of these harmonics with some noisy stochastic component.

To understand the intuition behind using the frequency spectrum as a source of compression we show in Figure 2 the power spectrum of two specific ports for a single week. In each case the power spectrum amplitudes are normalized with respect to the maximum amplitude frequency for that signal (usually the mean or 0^{th} frequency component), and the y-axis is shown on a log-scale after normalization. We observe that the power spectrum exhibits only a few very high energy components. For example the central peak and the high energy band around it corresponds to the mean (0^{th}) frequency in the Fourier decomposition, while the slightly lesser peaks symmetric around zero, and close to it correspond to the high energy frequencies that have a wavelength corresponding to the duration of a day.

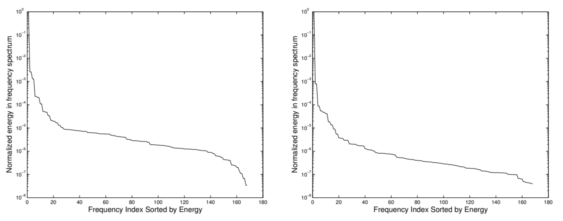


(a) Port 25/TCP (b) Port 4662/TCP

Figure 2: Frequency power spectrum of time-series of incoming flows on specific ports over a single week

We also show the how the normalized amplitude decreases when we sort the frequency components in descending order of their amplitudes in Figure 3. We observe that there is indeed a sharp drop (the figures are in log-scale on y-axis) in the energy of the frequency components after 20-30 components for the different signals considered.

We observe that a small number of components do capture a significant portion of the energy, which suggests a rather obvious compression scheme. For each week-long time series, pick the k frequencies that have the highest energies in the power spectrum. Figure 4 indicates that using 40 coefficients per week (around 40/168 = 25% of the original signal size) coefficients yields a relative error of less than 0.05 for more than 90% of all ports[3]. A relative error of 0.05 using our relative error metric indicates that around 95% of the original signal energy was captured in the compressed form. We observe in Figure 5 that the corresponding compressibility of UDP ports is slightly worse.



(a) Port 25/TCP (b) Port 4662/TCP

Figure 3: Energy of the frequencies sorted in descending order for specific ports

The reason is that the traffic volumes on UDP ports tend to exhibit far lesser aggregation, in terms of absolute volumes and popularity of usage of particular ports. Intuitively one expects that with higher volumes and aggregation levels, the traffic would exhibit more periodic structure, which explains the better compression for TCP ports as opposed to UDP ports.

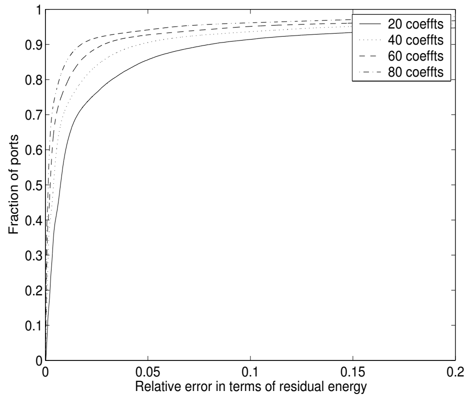


Figure 4: CDFs of relative error for TCP ports (incoming flows) with Fourier dictionary

The Fourier basis is one simple orthonormal basis. There are a host of other orthonormal bases which have been employed for compressing different datasets. Wavelets have traditionally been used for de-noising and compression in image and audio applications. The effectiveness of a wavelet basis depends on the choice of the "mother wavelet" function. However, identifying the best basis for representing either a given signal or a class of signals is a hard problem, for which only approximate answers exist using information-theoretic measures [17]. For our experiments we tried a variety of wavelet families including the well studied Daubechies family of wavelets, and other derivatives such as Symlets and Coiflets. Our observation is that the families of wavelets we tested had poorer performance when compared with the Fourier basis. Although an exhaustive discussion of choosing the ideal wavelet family is beyond the scope of this paper, our experiments with a host of wavelet families indicate that the traffic time-series cannot be efficiently compressed using wavelets (as an orthonormal basis) alone.

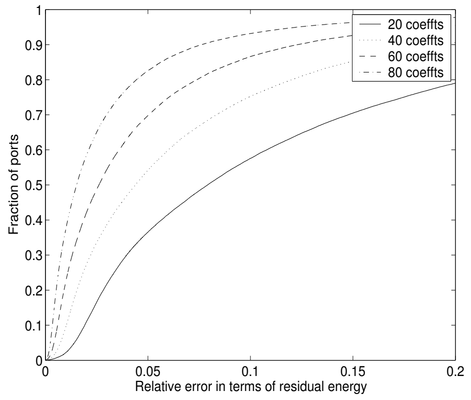


Figure 5: CDFs of relative error for UDP ports (incoming flows) with Fourier dictionary

6.2 Using Redundant Dictionaries

Our choice of the Fourier dictionary was motivated by the observation that the traffic time-series when viewed at a reasonable level of aggregation possesses a significant periodic component. Therefore, using Fourier basis functions as part of the redundant dictionary seems a reasonable starting point. There are however, other interesting incidents we wish to capture in the compressed representation. Experience with traffic data indicates that interesting events with high volume (and hence high signal energy) include possibly anomalous spikes, traffic dips, and slightly prolonged high traffic incidents. Such isolated incidents, localized in time, cannot be succinctly captured using only a Fourier basis. Fortunately, these events can be modeled either using spike functions appropriately placed at different time indices, or using Haar wavelets (square waveforms) of different scales and all translations. The fully-translational Haar wavelets at all scales and all translations form a rich redundant dictionary of size $N \log N$. By contrast, the orthonormal basis of Haar wavelets is of size N and consists of the Haar wavelets at all scales and only those translations which match the scale of the wavelet.

Table 1 compares a host of possible dictionaries on selected ports. Over the entire spectrum of port types, we observe that specific bases are indeed better suited than others for specific ports. For example, we observe that for some high volume and P2P ports using a Fourier dictionary gives better compression than using a wavelet or full-translation Haar dictionary, while for some of the random and scan ports, the wavelet or full-translation Haar dictionary give better compression. In some cases (e.g. port 114) we also find that using spikes in the dictionary gives the lowest compression error.

Rather than try to optimize the basis selection for each specific port, we wish to use redundant dictionaries that can best capture the different components that can be observed across the entire spectrum of ports. Hence we use redundant dictionaries composed of Fourier, fully-translational Haar, and Spike waveforms and observe that we can extract the best compression (in terms of number of coefficients selected), across an entire family of traffic time series data. We compare three possible redundant dictionaries: Fourier+ Haar wavelets (referred to as D_{F+H}), Fourier + Spikes (referred to as D_{F+S}), and Fourier + Spikes + Haar wavelets (referred to as D_{F+H+S}). Within each dictionary the error-compression tradeoff is determined by the number of coefficients chosen (Recall that a m-coefficient representation roughly corresponds to a compression ratio of m/N). A fundamental property of the greedy pursuit approach is that with every iteration the residual energy decreases, and hence the error is a monotonically decreasing function of the number of modes chosen. We evaluate the error-compression tradeoffs for these different dictionaries in Figures 6 and 7, where we assume that we are constrained to use 30 coefficients (roughly corresponding to using only one-sixth of the data points for each week). We observe two main properties of using the redundant dictionary approach. First, the compressibility is substantially enhanced by expanding the dictionary to include either spikes or Haar wavelets, in addition to the periodic Fourier components, i.e., using redundant dictionaries yields better fidelity for the same storage cost as compared to a single orthonormal basis. The second property we observe with the particular choice of basis functions on the traffic data is a monotonicity property – adding a richer basis set to the dictionary helps the compressibility. For example the error-compression tradeoff that results with D_{F+H+S} is never worse than either D_{F+H} or D_{F+S}. The compression does come at a slightly higher computation cost, since the time to compress the time series depends on the size of the dictionary used, as the compression time scales in linearly with the number of vectors in the dictionary (refer Section 4).

In Figures 8 and 9 we show how the 95th percentile of the relative error across all the ports decreases as a function of the number of coefficients used for representing the traffic data for each port for TCP and UDP ports respectively. We find that after 30-35 coefficients we gain little by adding additional coefficients, i.e., the marginal improvement in the fidelity of the representation becomes less significant. We will address this issue again in Section 8, by considering the rate of decrease of the residual as a function of the number of modes selected for specific ports, to derive

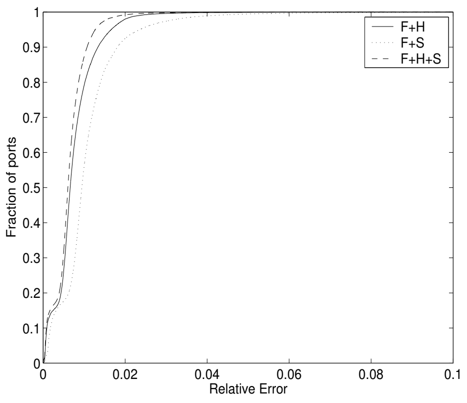


Figure 6: CDFs of relative error for TCP ports (incoming flows) with 30 coefficients for different dictionaries

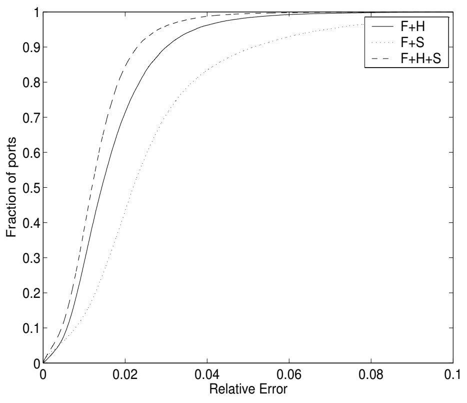


Figure 7: CDFs of relative error for UDP ports (incoming flows) with 30 coefficients for different dictionaries

stopping criteria for obtaining compressed representations.

6.3 Analysis of Selected Modes

We proceed to analyze the set of dictionary components that are chosen in the compressed representation using the redundant dictionaries for different ports, along different spatial and temporal dimensions. First, we are interested to see if there is substantial similarity in the set of dictionary components selected in the compressed representation across different ports. Second, we want to observe the temporal properties of compression; i.e., for a fixed traffic dimension, how does one week differ from another in terms of the components selected from the redundant dictionary? Third, we want to identify possible sources of correlation across the different traffic aggregates (flows, packets, bytes, both to and from) on a particular port of interest. Such analysis not only helps us to understand the nature of the underlying constituent components that make up each traffic

Table 1: Compression error with 30 coefficient representation for selected TCP ports (Legend: F = Fourier, W = Orthonormal $db4$ wavelets, H = Fully-translational Haar wavelets, S = Spikes)

Port Type	Port Number	Relative error with different dictionaries							
		D_F	D_W	D_S	D_H	D_{F+S}	D_{F+H}	D_{F+H+S}	D_{H+S}
High Volume	25	0.0005	0.0026	0.8446	0.0007	0.0004	0.0004	0.0004	0.0007
	80	0.0052	0.0256	0.7704	0.0074	0.0052	0.0018	0.0018	0.0073
P2P	1214	0.0003	0.0036	0.0007	0.8410	0.0003	0.0001	0.0001	0.0007
	6346	0.0009	0.0056	0.8193	0.0013	0.0009	0.0005	0.0005	0.0013
Scan	135	0.0016	0.0216	0.7746	0.0049	0.0015	0.0008	0.0008	0.0049
	9898	0.0066	0.0143	0.7800	0.0036	0.0063	0.0032	0.0032	0.0036
Random	5190	0.0023	0.0280	0.7916	0.0040	0.0023	0.0010	0.0010	0.0039
	114	0.5517	0.1704	0.0428	0.0218	0.0097	0.0218	0.0068	0.0068

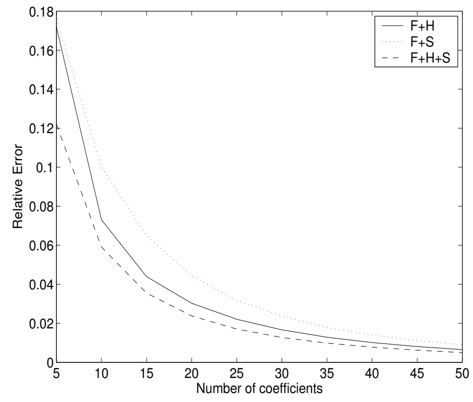


Figure 8: 95th percentile of relative error vs. number of coefficients selected for TCP ports (incoming flows)

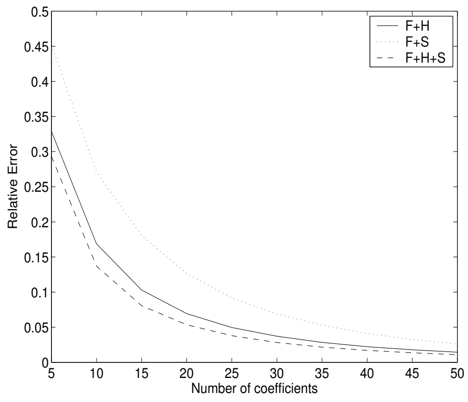


Figure 9: 95th percentile of relative error vs. number of coefficients selected for UDP ports (incoming flows)

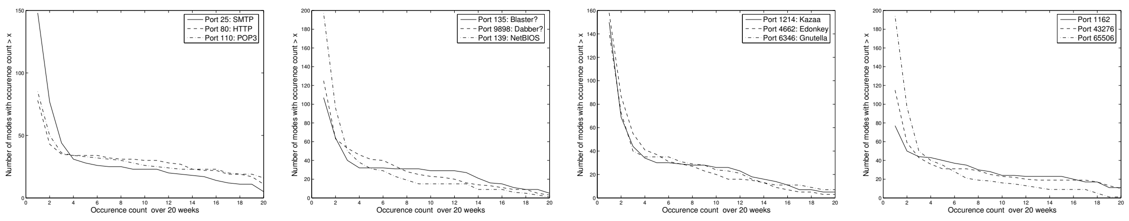
time series but also enables us to identify possible sources of joint compression, to further reduce the storage requirements. For the discussion presented in this section, we use the dictionary D_{F+S} (Fourier + Spike) as the redundant dictionary for our analysis.

6.3.1 Spatial Analysis Across Ports

We observe that the majority of selected dictionary components are restricted to a small number of ports—this is expected as these modes capture the minor variations across different ports, and also represent traffic spikes that may be isolated incidents specific to each port. We also observe that there are a few components that are consistently selected across almost all the ports. These components that are present across all the ports under consideration include the mean (zero-th Fourier component), the diurnal/off-diurnal periodic components, and a few other periodic components which were found to be the highest energy components in the Fourier analysis presented in Section 6.1.

6.3.2 Temporal Analysis Across Multiple Weeks

We also analyze, for specific instances of ports as defined by our four categories, the temporal stability of the set of components that are selected across different weeks over the 20 week data set, using 30 modes per week. As before, we use D_{F+S} as the redundant dictionary for compression. For each dictionary component (periodic component or spike) that is selected in the compressed representation over the 20 week period, we count the number of weeks in which it is selected. We show in Figure 10 the number of components that have an occurrence count more than x, as a function of x. We observe that the majority of the components are selected only for 1-2 weeks, which indicates that these captured subtle traffic variations from week to week. To further understand the stability of the components, we divide them into 3 categories: components that occur every week, components that occurred greater than 50% of the time (i.e, were selected 10-20 times over the 20 week period), and components that occurred fewer than 50% of the time (i.e., fewer than 10 times). Table 2 presents the break-



(a) High Volume Ports (b) Scan Target Ports (c) P2P Ports (d) Random Ports

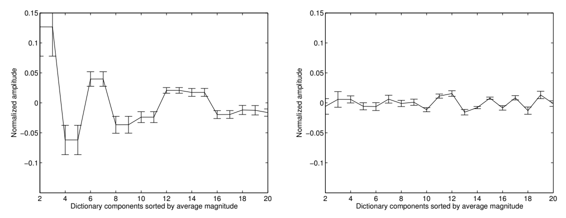
Figure 10: Occurrence counts using a 30 coefficient representation with D_{F+S}:Fourier+Spike over a 20 week period

down for the above classification for different ports in each category, and also shows the type of components that occur within each count-class. We find that across all the ports, the dictionary components that are always selected in the compressed representation correspond to periodic components such as the diurnal and off-diurnal frequencies.

The stability of a component depends not only on the fact that it was selected in the compressed representation, but also on the amplitude of the component in the compressed representation. Hence, we also analyze the amplitudes of the frequently occurring components (that occur greater than 50% of the time) across the 20 week dataset. Figures 11 and 12 show the mean and deviation of the amplitudes returned by the greedy pursuit procedure for these frequently occurring components. For clarity, we show the amplitudes of the real and imaginary part of the Fourier (periodic) components separately. For each port, we first sort the components according to the average magnitude (i.e, the energy represented by both the real and imaginary parts put together) over the 20 week period. We normalize the values of the average amplitude in both real and imaginary parts, and the deviations by the magnitude of the mean (or zero-th Fourier) component. We observe that the amplitudes are fairly stable for many Fourier components across the different port types. These results suggest that these stable (Fourier) frequencies may indeed form fundamental components of the particular traffic time series. The relative stability of amplitudes in the compressed representation also indicates that it may be feasible to build traffic models, that capture the fundamental variations in traffic, using the compressed representations.

6.3.3 Spatial Analysis Across Traffic Metrics

The last component of our analysis explores the similarity in the traffic data across different aggregates for a given port, within each week. One naturally expects a strong correlation between the number of flows, the number of packets, and the number of bytes for the same port, and also reasonable correlation between the total incoming volume



(a) Real part (b) Imaginary part

Figure 11: Stability of amplitudes of dictionary components selected – High volume: Port 80

and the total outgoing volume of traffic on the same port [4]. Figure 13 confirms this natural intuition about the nature of the traffic aggregates. We observe that for the high volume and P2P application ports, more than two-thirds of the dictionary components are commonly selected across all the different traffic aggregates and we also find that more than 30 components are selected across at least 4 of the traffic aggregates (bytes, packets, flows both to and from the port). We found that such similarity in the selected components across the different aggregates is less pronounced for the scan target ports and the random ports under consideration. Our hypothesis is that the distribution of packets per flow and bytes per packet are far more regular for the high volume applications (for example most HTTP, P2P packets use the maximum packet size to get maximum throughput) than on the lesser known ports (which may be primarily used as source ports in small sized requests).

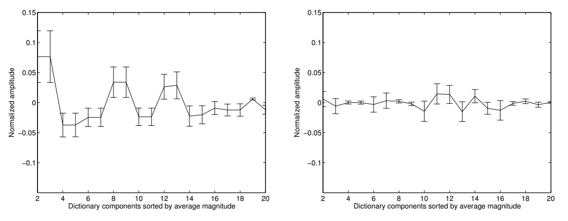
7 Applications

7.1 Visualization

One of the primary objectives of compression is to present to the network operator a high fidelity approximation that

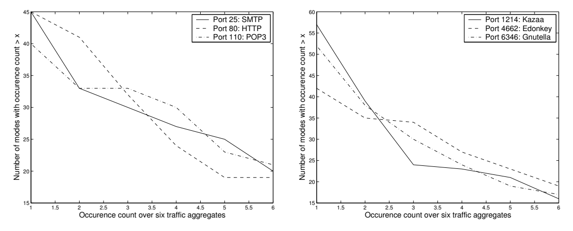
Table 2: Analyzing stable dictionary components for different classes of ports

Port Type	Port Number	All 20 weeks		10-20 weeks		0-10 weeks	
		Periodic	Spike	Periodic	Spike	Periodic	Spike
High Volume	25	5	0	18	0	23	102
	80	11	0	19	0	15	33
P2P	1214	5	0	21	0	20	104
	6346	7	0	17	0	23	94
Scan	135	5	0	24	0	15	63
	9898	3	0	20	0	35	67
Random	5190	11	0	10	0	27	73
	65506	1	0	15	0	31	147



(a) Real part (b) Imaginary part

Figure 12: Stability of amplitudes of dictionary components selected – P2P Port: 1214



(a) High Volume Ports (b) P2P Ports

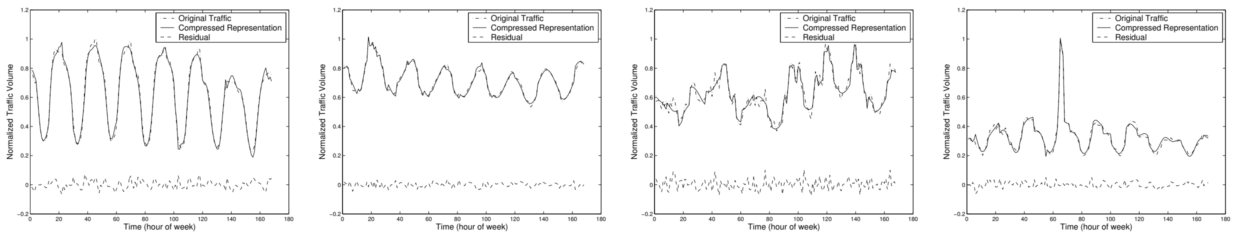
Figure 13: Occurrence counts using 30 coefficient representation with D_{F+S}:Fourier+Spike over different traffic aggregates for a single week

captures salient features of the original traffic metric of interest. Visualizing historical traffic patterns is a crucial aspect of traffic monitoring that expedites anomaly detection and anomaly diagnosis involving a network operator, who can use historical data as visual aids. It is therefore imperative to capture not only the periodic trends in the traffic, but also the isolated incidents of interest (for example, a post-lunch peak in Port 80 traffic, the odd spike in file sharing applications, etc).

Figure 14 shows some canonical examples from each of the four categories of ports we described earlier. In each case we show the original traffic time series over a week and the time series reconstructed from the compressed representation using 1:6 compression with D_{F+H+S}(Fourier + Haar + Spike). We also show the residual signal, which is the point-wise difference between the original signal and the compressed reconstruction. The traffic values are normalized with respect to the maximum traffic on that port observed for the week. We find that the compressed representations provide a high fidelity visualization of the original traffic data. Not surprisingly, the ports which exhibit the greatest amount of regularity in the traffic appear to be most easily compressible and the difference between the actual and compressed representation is almost negligible for these cases. It is also interesting to observe in each case that the compressed representation captures not only the periodic component of the signal, but also traffic spikes and other traffic variations.

7.2 Traffic Trend Analysis

Analyzing trends in traffic is a routine aspect in network operations. Operators would like to understand changes and trends in the application mix that is flowing through the network (e.g. detecting a a new popular file sharing protocol). Understanding traffic trends is also crucial for traffic engineering, provisioning, and accounting applications. It is therefore desirable that such trend analysis performed on the compressed data yields accurate results when compared to similar trend analysis on the raw (uncompressed) data. A simple method to extract trends over long timescales is to take the weekly average, and find a linear fit (using simple linear regression to find the slope of the line of best fit) to the weekly averages over multiple weeks of data. In Figure 15, we plot the relative error in estimating such a linear trend. We estimate the trend using 20 weeks of data for dif-



(a) Port 80 (b) Port 6346 (c) Port 9898 (d) Port 43726

Figure 14: Miscellaneous Ports using D_{F+H+S}: Fourier + Haar Wavelets + Spikes

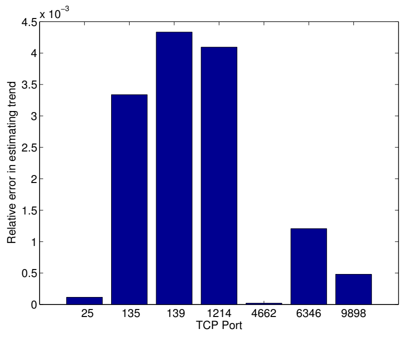


Figure 15: Relative error in estimating traffic trends

ferent ports, and in each case we estimate the slope of the best linear fit on the raw data and on the compressed data (using a 30 coefficient representation using D_{F+H+S}). We observe that across the different ports, the relative error in estimating the trend is less than 0.5%, which reaffirms the high fidelity of the compression techniques.

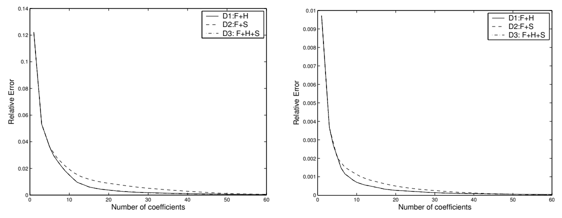
7.3 Modeling and Anomaly Detection

We observed in Section 6.3 that the underlying fundamental components are stable (both in terms of occurrence and their amplitudes) over time. It is conceivable that traffic models for anomaly detection can be learned on the compressed data alone. Our initial results suggest that traffic models [15] learned from compressed data have almost identical performance to models learned from uncompressed data, and hence compression does not affect the fidelity of traffic modeling techniques. Ongoing work includes evaluating different models for building prediction models for real-time anomaly detection using accurate yet parsimonious prediction models generated from the insights gained from the compression procedures.

8 Discussion

8.1 Stopping Criteria

In our experiments, we fixed the number of coefficients across all ports. One can imagine a host of stopping criteria to apply. One particularly interesting observation is that in many of the cases, a few of which are depicted in Figure 16, we find that the residual energy has a distinct knee beyond which the rate of drop in the residual energy is significantly lower. Intuitively one can imagine as the knee corresponding to the stochastic noise component of the original signal, which cannot be efficiently represented using any fundamental component. Note that the anomalous incidents such as spikes or glitches are usually captured before we hit the knee of the curve, as observed in Section 7.1. This raises the possibility that we have a *robust representation* of the original signal—one that does not change with the addition of noise as there are diminishing returns for any added effort aimed at modeling the noise component, which are not necessarily of interest either from a visualization or modeling perspective. We have



(a) Port 80 (b) Port 1214

Figure 16: Evaluating Stopping Criteria: Relative Error vs. number of coefficients

performed independent experiments with synthetic time series signals, similar to traffic time series (sinusoidal signals, with spikes and different noise patterns thrown in). We observe that in almost all the cases we observe a distinct knee in the redundant dictionary decomposition, once the fundamental high energy components get picked. We also find that the asymptotic slope of the curve of the residual energy beyond the knee has a unique signature that is characterized by the nature of the noise component (Gaussian or "White" vs. Power-law or "Colored"), and the redundant dictionary used.

8.2 Smaller Scales

At an appropriate aggregation level, network traffic will exhibit some periodicities. Traffic time series data from a variety of settings (enterprise and university) also confirm this hypothesis. These data typically represent the aggregate traffic at the border of a reasonably large network with fairly high aggregation levels. We believe that the methods for time-series compression using matching pursuit with redundant dictionaries are still applicable to data even at slightly lower scales of aggregation.

One of the objectives of compressing the time series is to enable different scales of time resolution for anomaly detection. It is imperative that the time scale for detecting traffic anomalies be less than the minimum time required for a large network attack to saturate. When the compression is applied to traffic aggregates at finer time granularities (e.g. for each week if we had volume counts for each five minute bin instead of hourly time bins), one expects that the effective compression would be better. The rationale behind the intuition arises from the fact that the high energy fundamental components correspond to relatively low frequency components, and such pronounced periodicities are unlikely to occur at finer time-scales. As a preliminary confirmation of this intuition, we performed the same compression procedures on a different data set, consisting of 5 minute traffic rates collected from SNMP data from a single link. Note that with 5-minute time intervals, we have $168 \times 12 = 2016$ data points per week. Figure 17 the relative error as a function of the number of coefficients used in the compressed representation (using D_{F+S}). We observe that with less than 40 (= 2% of the original space requirement) coefficients we are able to adequately compress the original time-series (with a relative error of less than 0.005), which represents significantly greater possible compression than those observed with the hourly aggregates.

8.3 Encoding Techniques

We observed that with larger dictionaries that include full-translation wavelets, we can achieve better compression. There is, however, a hidden cost in the effective compression with larger dictionaries as the indices of a larger dictionary potentially require more bits to represent than the indices of a smaller dictionary. One can imagine better ways of encoding the dictionary indices (e.g., using Huffman coding) to reduce the amount of space used up for storing the dictionary indices in addition to the component amplitudes. Our work explored the potential benefit of using signal processing methods for lossy compression and we observed that there is a substantial reduction in the storage requirement using just the methods presented in this paper. Many compression algorithms use lossy compression techniques along with efficient encoding techniques (lossless compression) to get the maximum compression gain, and such combinations of lossy and lossless compression methods can be explored further.

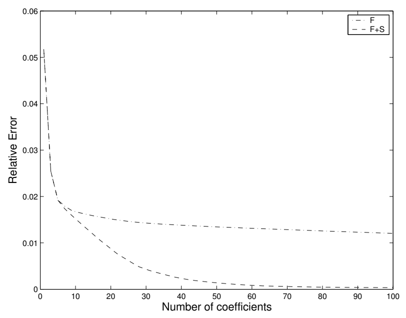


Figure 17: Compressing SNMP data collected at five minute intervals from a single link

8.4 Joint Compression

We observe that there are multiple sources of correlation across the different traffic dimensions that may be additionally utilized to achieve better compression. The temporal stability of the compressed representations (Section 6.3.2) suggests there is scope for exploiting the similarity across different weeks for the same traffic aggregate. For example, we could build a stable model over k weeks of data for each port/prefix and only apply the sparse approximation to the difference of each particular week from the model. Alternately one could imagine applying the simultaneous compression algorithms [16] across the different weeks for the same port. The simultaneous compression algorithms approximate all these signals at once using different linear combinations of the same elementary signals, while balancing the error in approximating the data against the total number of elementary signals that are used. We also observed that there is reasonable correlation in spatial dimensions, since the compressed representation of different traffic aggregates such as flows, packets, and bytes show significant similarity (Section 6.3.3).

The observations of the low dimensionality of network traffic data across different links also raises the possibility of using Principal Component Analysis (PCA) [10] for extracting better spatial compression, both across different traffic aggregates (e.g. different ports, across time) and across different measurements (e.g. across per-link, per-router counts). PCA like methods can be used to extract the sources of correlation before one applies redundant dictionary approaches to compress the traffic data. For example we can collapse the 20 week data set for a single port into a single matrix of traffic data, on which PCA like techniques can be applied to extract the first few common components, and the redundant dictionary can be applied on the residual (the projection on the non-principal subspace) to obtain a higher fidelity representation.

9 Conclusions

There is a pressing need for fine-grained traffic analysis at different scales and resolutions across space and time for network monitoring applications. Enabling such analysis requires the ability to store large volumes of historical data across different links, routers, and customers, for generating visual and diagnostic aids for network operators. In this paper, we presented a greedy pursuit approach over redundant dictionaries for compressing traffic time series data, and evaluated them using measurements from a large ISP. Our observations indicate that the compression models present a high fidelity representation for a wide variety of traffic monitoring applications, using less than 20% of the original space requirement. We also observe that most traffic signals can be compressed and characterized in terms of a few stable frequency components. Our results augur well for the visualization and modeling requirements for large scale traffic monitoring. Ongoing work includes evaluating and extracting sources of compression across other spatial and temporal dimensions, and evaluating the goodness of traffic models generated from compressed representations.

References

[1] Barford, P., Kline, J., Plonka, D., and Ron, A. A Signal Analysis of Network Traffic Anomalies. In *Proc. of ACM/USENIX Internet Measurement Workshop* (2002).

[2] Duffield, N. G., Lund, C., and Thorup, M. Charging From Sampled Network Usage. In *Proc. of ACM SIGCOMM Internet Measurement Workshop* (2001).

[3] Estan, C., Savage, S., and Varghese, G. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *Proc. of ACM SIGCOMM* (2003).

[4] Frossard, P., Vandergheynst, P., i Ventura, R. M. F., and Kunt, M. A posteriori quantization of progressive matching pursuit streams. *IEEE Trans. Signal Processing* (2004), 525–535.

[5] Gilbert, A. C., Muthukrishnan, S., and Strauss, M. J. Approximation of functions over redundant dictionaries using coherence. In *Proc. of 14th Annual ACM-SIAM Symposium on Discrete Algorithms* (2003).

[6] Gribonval, R., and Bacry, E. Harmonic decomposition of audio signals with matching pursuit. *IEEE Trans. Signal Processing* (2003), 101–111.

[7] Indyk, P. *High-dimensional computational geometry.* PhD thesis, Stanford University, 2000.

[8] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based Change Detection: Methods, Evaluation, and Applications. In *Proc. of ACM/USEINX Internet Measurement Conference* (2003).

[9] Lakhina, A., Crovella, M., and Diot, C. Diagnosing network-wide traffic anomalies. In *Proc. of ACM SIGCOMM* (2004).

[10] Lakhina, A., Papagiannaki, K., Crovella, M., Diot, C., Kolaczyk, E., and Taft, N. Structural analysis of network traffic flows. In *Proc. of ACM SIGMETRICS* (2004).

[11] Lempel, A., and Ziv, J. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory 24*, 5 (1978), 530–536.

[12] Mallat, S., and Zhang, Z. Matching pursuits with time frequency dictionaries. *IEEE Trans. Signal Processing 41*, 12 (1993), 3397–3415.

[13] Miller, A. J. *Subset selection in regression*, 2nd ed. Chapman and Hall, London, 2002.

[14] Cisco Netflow. `http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml`.

[15] Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., and Zhang, Y. Experience in measuring internet backbone traffic variability: Models, metrics, measurements and meaning. In *Proc. of International Teletraffic Congress (ITC)* (2003).

[16] Tropp, J. A., Gilbert, A. C., and Strauss, M. J. Algorithms for simultaneous sparse approximation part i: Greedy pursuit. *submitted* (2004).

[17] Zhuang, Y., and Baras, J. S. Optimal wavelet basis selection for signal representation. Tech. Rep. CSHCN TR 1994-7, Institute for Systems Research, Univ. of Maryland, 1994.

Notes

[1]Typically, k' is less than k, i.e. the magnitudes of the amplitudes of the dictionary components are less than the original time series.

[2]The slowest step in OMP is choosing the waveform which maximizes the dot product with the residual at each step. We can speed up this step with a Nearest Neighbors data structure [7] and reduce the time complexity for each iteration to $N + \mathrm{polylog}(d)$.

[3]Note that for each Fourier coefficient, we need to store both the real part and the imaginary part. It appears that we may actually need twice the space. However, the amplitudes for frequency f and frequency $-f$ are the same (except that they are complex conjugates of one another), we can treat them as contributing only two coefficients to the compressed representation together in total as opposed to four coefficients.

[4]We however note that there may be certain exceptional situations (e.g., worm or DDoS attacks that use substantially different packet and byte types) where such stable correlations between the flow, packet, and byte counts may not always hold.

Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic

Stefan Kornexl *TU München*   Vern Paxson *ICSI / LBNL*   Holger Dreger *TU München*   Anja Feldmann *TU München*   Robin Sommer *TU München*

Abstract

There are times when it would be extraordinarily convenient to record the entire contents of a high-volume network traffic stream, in order to later "travel back in time" and inspect activity that has only become interesting in retrospect. Two examples are security forensics—determining just how an attacker compromised a given machine—and network trouble-shooting, such as inspecting the precursors to a fault after the fault. We describe the design and implementation of a *Time Machine* to efficiently support such recording and retrieval. The efficiency of our approach comes from leveraging the heavy-tailed nature of network traffic: because the bulk of the traffic in high-volume streams comes from just a few connections, by constructing a filter that records only the first N bytes of each connection we can greatly winnow down the recorded volume while still retaining both small connections in full, and the beginnings of large connections (which often suffices).

The system is designed for operation in Gbps environments, running on commodity hardware. It can hold a few minutes of a high volume stream in RAM, and many hours to days on disk; the user can flexibly configure its operation to suit the site's nature. We present simulation and operational results from three distinct Gbps production environments exploring the feasibility and efficiency of a Time Machine implementation. The system has already proved useful in enabling analysis of a break-in at one of the sites.

1 Introduction

Network packet traces—particularly those with not only headers but full contents—can prove invaluable both for trouble-shooting network problems and for investigating security incidents. Yet in many operational environments the sheer volume of the traffic makes it infeasible to capture the entire stream or retain even significant subsets for extended amounts of time. Of course, for both troubleshooting and security forensics, only a very small proportion of the traffic actually turns out to be pertinent. The problem is that one has to decide *beforehand*, when configuring a traffic monitor, what context will turn out to be relevant *retrospectively* to investigate incidents.

Only in low volume environments can one routinely bulk-record all network traffic using tools such as `tcpdump` [2]. Rising volumes inevitably require filtering. For example, at the *Lawrence Berkeley National Laboratory* (LBNL), a medium-size Gbps environment, the network traffic averages 1.5 TB/day, right at the edge of what can be recorded using commodity hardware. The site has found it vital to record traffic for analyzing possible security events, but cannot retain the full volume. Instead, the operators resort to a `tcpdump` filter with *85 terms* describing the traffic to skip—omitting any recording of key services such as HTTP, FTP data, X11 and NFS, as well as skipping a number of specific high-volume hosts, and all non-TCP traffic. This filter reduces the volume of recorded traffic to about 4% of the total.

At higher traffic rates, even such filtering becomes technically problematic. For example, the *Munich Scientific Research Network* (Münchner Wissenschaftsnetz, MWN), a heavily-loaded Gbps university environment, averages more than 2 TB external traffic each day, with busy-hour loads of 350 Mbps. At that level, it is very difficult to reliably capture the full traffic stream using a simple commodity deployment.

A final issue concerns *using* the captured data. In cases of possible security compromise, it can be of great importance to track down the attacker and assess the damage as quickly as possible. Yet, manually sifting through an immense archive of packet traces to extract a "needle in a haystack" is time-consuming and cumbersome.

In this work we develop a system that uses dynamic packet filtering and buffering to enable effective bulk-recording of large traffic streams. As this system allows us to conveniently "travel back in time", we term it a *Time Machine*. Our Time Machine buffers network streams first in memory and then on disk, providing several days of nearly-complete (from a forensics and trouble-shooting perspective) historic data and supporting timely access to locate the haystack needles. Our initial application of the Time Machine is as a forensic tool, to extract detailed past information about unusual activities once they are detected. Already the Time Machine has proved operationally useful, enabling diagnosis of a break-in that had gone overlooked at LBNL, whose standard bulk-recorder's static filter had missed capturing the relevant data.

Naturally, the Time Machine cannot buffer an entire high-volume stream. Rather, we exploit the "heavy-tailed" nature of network traffic to partition the stream more effectively (than a static filter can) into a small subset of high interest versus a large remainder of low interest. We then record the small subset and discard the rest. The key insight that makes this work is that most network connections are quite short, with only a small number of large connections (the heavy tail) accounting for the bulk of the total volume [6]. However, very often for forensics and trouble-shooting applications the *beginning* of a large connection contains the most significant information. Put another way, given a choice between recording some connections in their entirety, at the cost of missing others in their entirety; versus recording the beginnings of all connections and the entire contents of most connections, we generally will prefer the latter.

The Time Machine does so using a *cutoff* limit, N: for every connection, it buffers up to the first N bytes of traffic. This greatly reduces the traffic we must buffer while retaining full context for small connections and the beginning for large connections. This simple mechanism is highly efficient: for example, at LBNL, with a cutoff of $N = 20$ KB and a disk storage budget of 90 GB, we can retain 3–5 days of *all* of the site's TCP connections, and, using another 30 GB, 4–6 days for all of its UDP flows (which tend to be less heavy-tailed).

We are not aware of any comparable system for traffic capture. While commercial bulk recorders are available (e.g, *McAfee Security Forensics* [3]), they appear to use brute-force bulk-recording, requiring huge amounts of disk space. Moreover, due to their black-box nature, evaluating their performance in a systematic fashion is difficult. Another approach, used by many network intrusion detection/prevention systems, is to record those packets that trigger alerts. Some of these systems buffer the start of every connection for a short time (seconds) and store them permanently if the session triggers an alert. Such systems do not provide long-term buffers or arbitrary access, so they do not support retrospective analysis of a problematic host's earlier activity. The Bro NIDS [5] can either record *all* analyzed packets, or *future* traffic once an incident has been detected. Finally, the Packet Vault system was designed to bulk record entire traffic streams [1]. It targets lower data rates and does not employ any filtering.

We organize the remainder of the paper as follows. In § 2, we briefly summarize the Time Machine's design goals. In § 3, we use trace-driven simulation to explore the feasibility of our approach for data-reduction in three high-volume environments. We discuss the Time Machine's architecture in § 4 and present an evaluation of its performance in two of the environments in § 5. § 6 summarizes our work.

2 Design Goals

We identified six major design goals for a Time Machine:

Provide raw packet data. The Time Machine should enable recording and retrieval of full packets, including payload, rather than condensed versions (e.g., summaries, or just byte streams without headers), in order to prevent losing crucial information.

Buffer traffic comprehensively. The Time Machine should manage its stored traffic for time-frames of *multiple days*, rather than seconds or minutes. It should not restrict capture to individual hosts or subnetworks, but keep as widespread data as possible.

Prioritize traffic. Inevitably, in high-volume environments we must discard some traffic quickly. Thus, the Time Machine needs to provide means by which the user can express different classes of traffic and the resources associated with each class.

Automated resource management. From experience, we know that having to manually manage the disk space associated with high-volume packet tracing becomes tedious and error-prone over time. The Time Machine needs to enable the user to express the resources available to it in high-level terms and then manage these resources automatically.

Efficient and flexible retrieval. The Time Machine must support timely queries for different subsets of the buffered data in a flexible and efficient manner. However, its packet capture operation needs to have priority over query processing.

Suitable for high-volume environments using commodity hardware. Even though we target large networks with heavily loaded Gbps networks, there is great benefit in a design that enables the Time Machine to run on off-the-shelf hardware, e.g., PCs with 2 GB RAM and 500 GB disk space.

3 Feasibility Study

In this section we explore the feasibility of achieving the design goals outlined above by leveraging the heavy-tailed nature of traffic to exclude most of the data in the high-volume streams.

Methodology: To evaluate the memory requirements of a Time Machine, we approximate it using a packet-buffer model. We base our evaluation on connection-level logs from the three environments described below. These logs capture the nature of their environment but with a relatively low volume compared to full packet-level data. Previous work [7] has shown that we can use flow data to approximate the data rate contributed by a flow, so we can assume that a connection spreads its total traffic across its duration evenly, which seems reasonable for most connections, especially large ones.

We evaluate the packet-buffer model in discrete time steps, enabling us to capture at any point the *volume* of packet data currently stored in the buffer and the *growth-rate* at which that volume is currently increasing. In our simplest simulation, the arrival of a new connection increases the growth-rate by the connection's overall rate (bytes transferred divided by duration); it is decreased by the same amount when it finishes. We then add the notion of keeping data for an extended period of time by introducing an *eviction time* parameter, T_e, which defines how long the buffer stores each connection's data. In accordance with our goals, we aim for a value of T_e on the order of days rather than minutes.

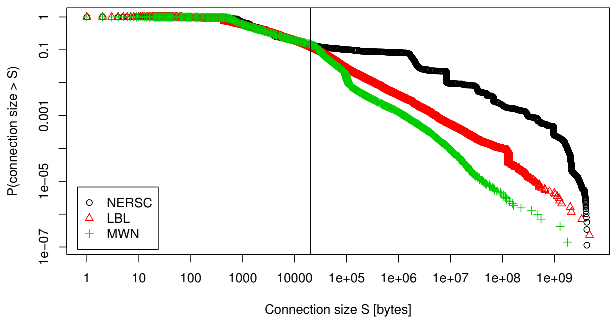


Figure 1: Log-log CCDF of connection sizes

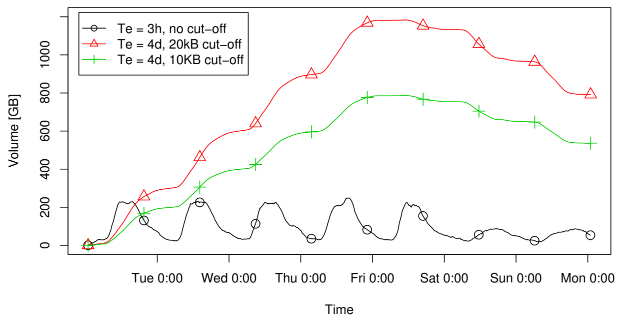


Figure 2: Simulated Volume for MWN environment

As described so far, the model captures bulk-recording with a timeout but without a *cutoff*. We incorporate the idea of recording only the first N bytes for each connection by adjusting the time at which we decrement the growth-rate due to each connection, no longer using the time at which the connection finishes, but rather the time when it exceeds N bytes (the connection *size cutoff*).

Environments: We drive our analysis using traces gathered from packet monitors deployed at the Internet access links of three institutions. While all institutions transfer large volumes of data (one to several TBs a day), their networks and traffic composition have qualitative differences.

MWN: The *Munich Scientific Research Network* (Münchner Wissenschaftsnetz, MWN) in Munich, Germany, connects two major universities and affiliated research institutions to the Internet, totaling approximately 50,000 hosts. The volume transferred over its Gbps Internet link is around 2 TB a day. Roughly 15–20% of the traffic comes from a popular FTP mirror hosted by one of the universities. The average utilization during busy-hours is about 350 Mbps (68 Kpps).

LBNL: The *Lawrence Berkeley National Laboratory* (LBNL) network in California, USA, comprises 9,000 hosts and 4,000 users, connecting to the Internet via a Gbps link with a busy-hour load of 320 Mbps (37 Kpps).

NERSC: The National Energy Research Scientific Computing Center is administratively part of LBNL, but physically separate and uses a different Internet access link; it provides computational resources (around 600 hosts) to 2,000 users. The traffic is dominated by large transfers, containing significantly fewer user-oriented applications such as the Web. The busy-hour utilization of the Gbps link is 260 Mbps (43 Kpps).

For our analysis we use connection-level logs of one week from MWN, LBNL, and NERSC. The MWN connection log contains 355 million connections from Monday, Oct. 18, 2004, through the following Sunday. The logs from LBNL and NERSC consist of 22 million and 4 million connections observed in the week after Monday Feb. 7, 2005 and Friday Apr. 29, 2005 respectively.

Analysis of connection size cutoff: As a first step we investigate the heavy-tailed nature of traffic from our environments. Figure 1 plots the (empirical) complementary cumulative distribution function (CCDF) of the number of bytes per connection for each of the three environments. Note that a "linear" relationship in such a log-log scaled plot indicates consistency of the tail with a Pareto distribution.

An important consideration when examining these plots is that the data we used—connection summaries produced by the Bro NIDS—are based on the difference in sequence numbers between a TCP connection's SYN and FIN packets. This introduces two forms of *bias*. First, for long-running connections, the NIDS may miss either the initial SYN or the final FIN, thus not reporting a size for the connection. Second, if the connection's size exceeds 4 GB, then the sequence number space will *wrap*; Bro will report only the bottom 32 bits of the size. Both of these biases will tend to *underestimate* the heavy-tailed nature of the traffic, and we know they are significant because the total traffic volume accounted for by the Bro reports is much lower than that surmised via random sampling of the traffic.

The plot already reveals insight about how efficiently a cutoff can serve in terms of reducing the volume of data the Time Machine must store. For a cutoff of 20 KB, corresponding to the vertical line in Figure 1, 12% (LBNL), 14% (NERSC) and 15% (MWN) of the connections have a larger total size. The percentage of bytes is much larger, though: 87% for MWN, 96% for LBNL, and 99.86% for NERSC. Accordingly, we can expect a huge benefit from using a cutoff.

Next, using the methodology described above we simulated the packet buffer models based on the full connection logs. Figures 2, 3 and 4 show the required memory for MWN, LBNL, and NERSC, respectively, for different combinations of eviction time T_e and cutoff. A deactivated cutoff corresponds to bulk-recording with a timeout. While the bulk-recording clearly shows the artifacts of time of day and day of week variations, using a cutoff reduces this effect, because we can accompany the cutoff with a much larger timeout, which spreads out the variations. We see that a cutoff of 20 KB quite effectively reduces the buffered volume: at LBNL, with $T_e = 4$ d, the maximum volume, 68 GB, is just a tad higher than the maximum volume, 64 GB, for bulk-recording with $T_e = 3$ h. However, we have increased the duration of data availabil-

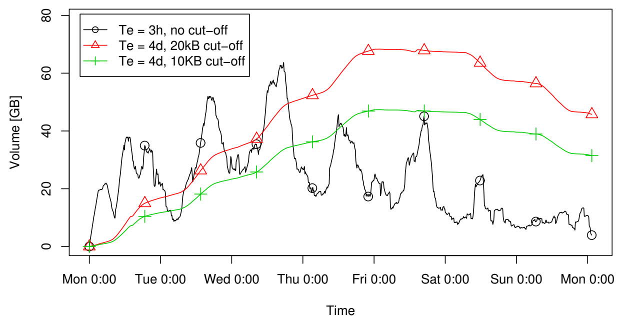


Figure 3: Simulated volume for LBNL environment

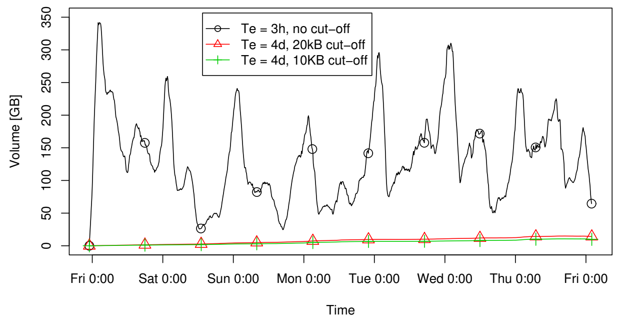


Figure 4: Simulated volume for NERSC environment

ity by a factor of 32! Note that the volume for simulations with $T_e = 4$ d stops to increase steadily after four days, since starting then connections are being evicted in the buffer model. At NERSC, the mean (peak) even decreases from 135 GB (344 GB) to 7.7 GB (14.9 GB). This enormous gain is due to the site's large proportion of high-volume data transfers. As already indicated by the lower fraction of bytes in the larger connections for MWN, the gain from the cutoff is not quite as large, likely due to the larger fraction of HTTP traffic.

Reducing the cutoff by a factor of two further reduces the maximum memory requirements, but only by a factor 1.44 for LBNL, 1.40 for NERSC, and 1.50 for MWN—not by a full factor of two. This is because at this point we are no longer able to further leverage a heavy tail.

The Figures also show that without a cutoff, the volume is spiky. In fact, at NERSC the volume required with $T_e = 1$ h is no more than two times that with $T_e = 1$ m, due to its intermittent bursts. On the other hand, with a cutoff we do not see any significant spikes in the volumes. This suggests that sudden changes in the buffer's growth-rate are caused by a few high-volume connections rather than shifts in the overall number of connections. All in all, the plots indicate that by using a cutoff of 10–20 KB, buffering *several days* of traffic is practical.

4 Architecture

The main functions our Time Machine needs to support are *(i)* buffering traffic using a cutoff, *(ii)* migrating (a subset of) the buffered packets to disk and managing the associated storage, *(iii)* providing flexible retrieval of subsets of the packets, and *(iv)* enabling customization. To do so, we use the multi-threaded architecture shown in Figure 5, which separates *user interaction* from *recording* to ensure that packet capture has higher priority than packet retrieval.

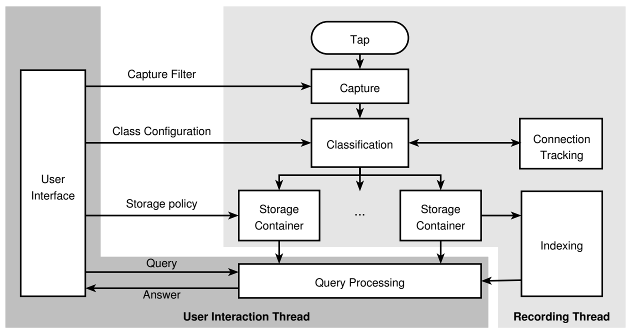


Figure 5: Time Machine System Architecture

The user interface allows the user to configure the recording parameters and issue queries to the *query processing* unit to retrieve subsets of the recorded packets. The recording thread is responsible for packet capture and storage. The architecture supports customization by splitting the overall storage into several *storage containers*, each of which is responsible for storing a subset of packets within the resources (memory and disk) allocated via the user interface. The *classification* unit decides which packets to assign to each storage container. In addition, the classification unit is responsible for monitoring the cutoff with the help of the *connection tracking* component, which keeps per connection statistics. To enable efficient retrieval, we use an index across all packets stored in all storage containers, managed by the *indexing* module. Finally, access to the packets coming in from the network *tap* is managed by the *capture* unit.

The capture unit receives packets from the network tap and passes them on to the classification unit. Using the connection tracking mechanism, it checks if the connection the packet belongs to has exceeded its cutoff value. If not, it finds the associated storage container, which then stores the packet in memory, indexing it in the process for quick access later on. It later migrates it to disk, and eventually deletes it. Accordingly, the actual Time Machine differs from the connection-level simulation model in that now the buffers are caches that evict *packets* when they are full, rather than evicting whole connections precisely at their eviction time.

Our implementation of the architecture uses the `libpcap` packet capture library [2], for which the user can specify a kernel-level BPF [4] capture filter to discard "uninteresting" traffic as early as possible. We collect and store each packet's full content and capture timestamp.

The capture unit passes the packet to the classification routines, which divide the incoming packet stream into classes according to a user-specified configuration. Each class definition includes a class name, a BPF filter to identify which packets belong to the class, a matching priority, and several storage parameters; for example:

```
class "telnet" { filter "tcp port 23";
  precedence 50; cutoff 10m;
  mem 10m; disk 10g; }
```

which defines a class "`telnet`" that matches, with priority 50, any traffic captured by the BPF filter `"tcp port 23"`. A cutoff of 10 MB is applied, and an in-memory buffer of 10 MB and a disk budget of 10 GB allocated.

For every incoming packet, we look up the class associated with its connection in the connection tracking unit, or, if it is a new connection, match the packet against all of the filters. If more than one filter matches, we assign it to the class with the highest priority. If no filter matches, the packet is discarded.

To track connection cutoffs, the Time Machine keeps state for all active connections in a hash table. If a newly arrived packet belongs to a connection that has exceeded the cutoff limit configured for its class, it is discarded. We manage entries in the connection hash table using a user-configurable inactivity timeout; the timeout is shorter for connections that have not seen more than one packet, which keeps the table from growing too large during scans or denial of service attacks.

For every class, the Time Machine keeps an associated storage container to buffer the packets belonging to the class. Storage containers consist of two ring buffers. The first stores packets in a RAM buffer, while the second buffers packets on disk. The user can configure the size of both buffers on a per-class basis. (A key motivation for maintaining a RAM buffer in addition to disk storage is to enable near-real-time access to the more recent part of the Time Machine's archive.) Packets evicted from the RAM buffer are moved to the disk buffer. We structure the disk buffer as a set of files. Each such file can grow up to a configurable size (typically 10–100s of MB). Once a file reaches this size, we close it and create a new file. We store packets both in memory and on disk in `libpcap` format. This enables easy extraction of `libpcap` traces for later analysis.

To enable quick access to the packets, we maintain *multiple* indexes. The Time Machine is structured internally to support any number of indexes over an arbitrary set of (predefined) protocol header fields. For example, the Time Machine can be compiled to simultaneously support per-address, per-port, and per-connection-tuple indexes. Each index manages a list of time intervals for every unique key value, as observed in the protocol header field (or fields) of the packets. These time intervals provide information on whether packets with that key value are available in a given storage container and at what starting timestamp, enabling fast retrieval of packets. Every time the Time Machine stores a new packet it updates each associated index. If the packet's key—a header field or combination of fields—is not yet in the index, we create a new entry containing a zero-length time interval starting with the timestamp of the packet. If an entry exists, we update it by either extending the time interval up to the timestamp of the current packet, or by starting a new time interval, if the time difference between the last entry in the existing interval and the new timestamp exceeds a user-defined parameter. Thus, this parameter trades off the size of the index (in terms of number of intervals we maintain) for how precisely a given index entry localizes the packets of interest within a given storage container. As interval entries age, we migrate them from in-memory index structures to index files on disk, doing so at the same time the corresponding packets in the storage containers migrate from RAM to disk. In addition, the user can set an upper limit for the size of the in-memory index data structure.

The final part of the architecture concerns how to find packets of interest in the potentially immense archive. While this can be done using brute force (e.g., running `tcpdump` over all of the on-disk files), doing so can take a great deal of time, and also have a deleterious effect on Time Machine performance due to contention for the disk. We address this issue using the query-processing unit, which provides a flexible language to express queries for subsets of the packets. Each query consists of a logical combination of time ranges, keys, and an optional BPF filter. The query processor first looks up the appropriate time intervals for the specified key values in the indexing structures, trimming these to the time range of the query. The logical *or* of two keys is realized as the union of the set of intervals for the two keys, and an *and* by the intersection. The resulting time intervals correspond to the time ranges in which the queried packets originally arrived. We then locate the time intervals in the storage containers using binary search. Since the indexes are based on time intervals, these only limit the amount of data that has to be scanned, rather then providing exact matches; yet this narrowing suffices to greatly reduce the search space, and by foregoing exact matches we can keep the indexes much smaller. Accordingly, the last step consists of scanning all packets in the identified time ranges and checking if they match the key, as well as an additional BPF filter if supplied with the query, writing the results to a `tcpdump` trace file on disk.

5 Evaluation

To evaluate the Time Machine design, we ran an implementation at two of the sites discussed in § 3. For LBNL, we used three classes, each with a 20 KB cutoff: TCP traffic, with a space budget of 90 GB; UDP, with 30 GB; and Other, with 10 GB. To evaluate the "hindsight" capabilities, we determine the *retention*, i.e., the distance back in time to which we can travel at any particular moment, as illustrated in Figure 6. Note how retention increases after the Time Machine starts until the disk buffers have filled. After this

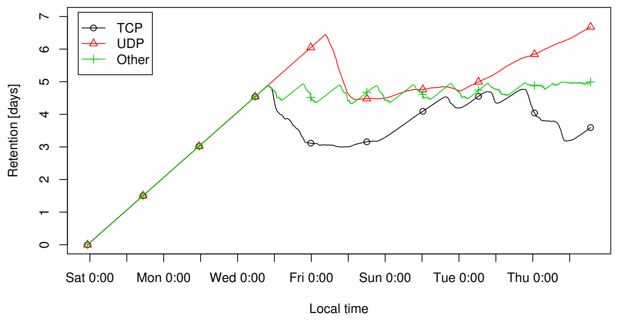


Figure 6: Retention in the LBNL environment

point, retention correlates with the incoming bandwidth for each class and its variations due to diurnal and weekly effects. New data forces the eviction of old data, as shown for example by the retention of TCP shortening as the lower level weekend traffic becomes evicted around Wed–Thu. The TCP buffer of 90 GB allows us to retain data for 3–5 days, roughly matching the predictions from the LBNL simulations (recall the volume biases of the connection-level data discussed in § 3). Use of a cutoff is highly efficient: on average, 98% of the traffic gets discarded, with the remainder imposing an average rate of 300 KB/s and a maximum rate of 2.6 MB/s on the storage system. Over the 2 weeks of operation, `libpcap` reported only 0.016% of all packets dropped.

Note that classes do not have to be configured to yield an identical retention time. The user may define classes based on their view of utility of having the matching traffic available in terms of cutoff and how long to keep it. For example we might have included a class configuration similar to the example in § 4 in order to keep more of Telnet connections for a longer period of time.

Operationally, the Time Machine has already enabled the diagnosis of a break-in at LBNL by having retained the response to an HTTP request that was only investigated three days later. The Time Machine's data both confirmed a successful compromise and provided additional forensic information in terms of the attacker's other activities. Without the Time Machine, this would not have been possible, as the site cannot afford to record its full HTTP traffic for any significant length of time.

At MWN we ran preliminary tests of the Time Machine, but we have not yet evaluated the retention capability systematically. First results show that about 85% of the traffic gets discarded, with resulting storage rates of 3.5 (13.9) MB/s average (maximum). It appears that the larger volume of HTTP traffic is the culprit for this difference compared to LBNL, due to its lesser heavy-tailed nature; this matches the results of the MWN connection-level simulation. For this environment it seems we will need to more aggressively exploit the classification and cutoff mechanisms to appropriately manage the large fraction of HTTP traffic.

The fractions of discarded traffic for both LBNL and MWN match our predictions well, and the resulting storage rates are reasonable for today's disk systems, as demonstrated in practice. The connection tracking and indexing mechanisms coped well with the characteristics of real Internet traffic. We have not yet evaluated the Time Machine at NERSC, but the simulations promise good results.

6 Summary

In this paper, we introduce the concept of a *Time Machine* for efficient network packet recording and retrieval. The Time Machine can buffer several days of raw high-volume traffic using commodity hardware. It provides an efficient query interface to retrieve the packets in a timely fashion, and automatically manages its available storage. The Time Machine relies on the simple but crucial observation that due to the "heavy-tailed" nature of network traffic, we can record most connections in their entirety, yet skip the bulk of the total volume, by storing up to (a customizable) cutoff limit of bytes per connection. We have demonstrated the effectiveness of the approach using a trace-driven simulation as well as operational experience with the actual implementation in two environments. A cutoff of 20 KB increases data availability from several hours to several days when compared to brute-force bulk recording.

In operational use, the Time Machine has already proved valuable by enabling diagnosis of a break-in that standard bulk-recording had missed. In future work, we intend to add a remote access interface to enable real-time queries for historic network data by components such as network intrusion detection systems.

7 Acknowledgments

This work was supported by the National Science Foundation under grant STI-0334088, and by a grant from the Bavaria California Technology Center, for which we are grateful.

References

[1] ANTONELLI, C., UNDY, M., AND HONEYMAN, P. The Packet Vault: Secure Storage of Network Data. In *Proc. Workshop on Intrusion Detection and Network Monitoring* (April 1999), pp. 103–110.

[2] LAWRENCE BERKELEY NATIONAL LABORATORY. tcpdump and libpcap. `http://www.tcpdump.org/`.

[3] MCAFEE. McAfee Security Forensics. `http://www.mcafeesecurity.com/us/products/mcafee/forensics/security_for%ensics.htm`.

[4] MCCANNE, S., AND JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proc. USENIX Winter 1993 Conference* (January 1993), pp. 259–270.

[5] PAXSON, V. Bro: A system for detecting network intruders in real-time. *Computer Networks 31*, 23–24 (December 1999).

[6] PAXSON, V., AND FLOYD, S. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), 226–224.

[7] WALLERICH, J., DREGER, H., FELDMANN, A., KRISHNAMURTHY, B., AND WILLINGER, W. A Methodology for Studying Persistency Aspects of Internet Flows. *ACM SIGCOMM Computer Communication Review 35*, 2 (April 2005), 23–36.

Improving Sketch Reconstruction Accuracy Using Linear Least Squares Method

Gene Moo Lee, Huiya Liu, Young Yoon and Yin Zhang
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
{gene,huiyaliu,agitato7,yzhang}@cs.utexas.edu

Abstract

Sketch is a sublinear space data structure that allows one to approximately reconstruct the value associated with any given key in an input data stream. It is the basis for answering a number of fundamental queries on data streams, such as range queries, finding quantiles, frequent items, etc. In the networking context, sketch has been applied to identifying heavy hitters and changes, which is critical for traffic monitoring, accounting, and network anomaly detection.

In this paper, we propose a novel approach called *lsquare* to significantly improve the reconstruction accuracy of the sketch data structure. Given a sketch and a set of keys, we estimate the values associated with these keys by constructing a linear system and finding the optimal solution for the system using linear least squares method. We use a large amount of real Internet traffic data to evaluate *lsquare* against *countmin*, the state-of-the-art sketch scheme. Our results suggest that given the same memory requirement, *lsquare* achieves much better reconstruction accuracy than *countmin*. Alternatively, given the same reconstruction accuracy, *lsquare* requires significantly less memory. This clearly demonstrates the effectiveness of our approach.

1 Introduction

For many network management applications, it is essential to accurately monitor and analyze network traffic. For example, Internet service providers need to monitor the usage information in order to support usage-based pricing. Network operators need to observe the traffic pattern to perform traffic engineering. Network anomaly detection systems need to continuously monitor the traffic in order to uncover anomalous traffic patterns in near real-time, especially those caused by flash crowds, denial-of-service attacks (DoS), worms, and network element failures. These applications typically treat the traffic as a collection of *flows* with some properties to keep track of (*e.g.*, volume, number of packets). The flows are typically identified by certain combination of packet header fields (*e.g.*, IP addresses, port numbers, and protocol).

A naïve approach for network traffic measurement is to maintain state and perform analysis on a *per-flow* basis. However, as link speeds and the number of flows increase, keeping per-flow state can quickly become either too expensive or too slow. As a result, a lot of recent networking research efforts have been directed towards developing scalable and accurate techniques for performing traffic monitoring and analysis without keeping per-flow state (*e.g.*, [6]). Meanwhile, computation over massive data streams has been an active research area in the database research community over the past several years. The emerging field of *data stream computation* deals with various aspects of computation that can be performed in a space- and time-efficient manner when each item in a data stream can be accessed only once (or a small number of times). A rich body of algorithms and techniques have been developed. A good survey of the algorithms and applications in data stream computation can be found in [11].

A particularly powerful technique is *sketch* [1, 7, 3, 5], a probabilistic summary data structure proposed for analyzing massive data streams. Sketches avoid keeping per-flow state by dimensionality reduction techniques, using projections along random vectors. Sketches have some interesting properties that have proven to be very useful in analyzing data streams: they are space efficient, provide provable probabilistic reconstruction accuracy guarantees, and are linear (*i.e.*, sketches can be combined in an arithmetical sense). These properties have made sketch the basis for answering a number of fundamental queries on data streams, such as range queries, finding quantiles and frequent items [11]. In the networking context, sketch has been successfully applied to detecting heavy hitters and changes [8, 4].

A key operation on the sketch data structure is so called *point estimation*, *i.e.*, to estimate the accumulated value associated with a given key. All existing methods perform point estimation for different keys separately and only have limited accuracy. In this paper, we propose a novel method called *lsquare* to significantly improve the accuracy of point estimation on the sketch data structure. Instead of estimating values for individual keys separately, *lsquare* first extracts a set of keys that is a superset of all the heavy hitter flows and then simultaneously estimates the accumulated values for this set of keys – it does so by first constructing a linear system and then finding the optimal solution to the system through linear least squares method.

We use a large amount of real Internet traffic data to evaluate our method against *countmin* [5], the best existing sketch scheme. Our results are encouraging: Given the same memory requirement, *lsquare* yields much more accurate estimates than *countmin*; and given the same reconstruction accuracy, *lsquare* uses significantly less memory.

The remainder of the paper is organized as follows. In Section 2, we give an overview of sketch data structure, define the problem, and survey the related work. In Section 3, we describe our *lsquare* method for point estimation on the sketch data structure. In Section 4, we evaluate the proposed method using real Internet traffic data. We conclude in Section 5.

2 Background

This section provides some background on the problem we want to solve. First, we briefly describe the underlying data stream model and the sketch data structure. Then we define the problem of point estimation on sketch and explain the existing methods to solve the problem. We will also briefly survey the related work.

2.1 Data Stream Model

Let $\mathcal{I} = (k_1, u_1), (k_2, u_2), \ldots$ be an input stream that arrives sequentially, item by item. Here $k_t \in \{0, \ldots, n-1\}$ is a key and $u_t \geq 0$ is the update value associated with the key. Let U_k be the sum of update values for a key k. Here, the update values are non-negative, meaning that U_k always increase. This model is called the *cash register model* [11]. Many applications of sketches guarantee that counts are non-negative. However, we note that our proposed method is also applicable to the more general *Turnstile model* [11], in which update values may be negative.

2.2 Count-Min Sketch

Sketch [5, 8, 14] is a sublinear space data structure for summarizing massive data streams. We use the notations in Table 1 to specify the sketch data structure.

Data structure: A *sketch* is a two-dimensional count array $T[i][j]$ $(0 \leq i < H, 0 \leq j < K)$, where H is the number of one-dimensional arrays and K is the number of counts in each array. Each count of sketch is initially set to zero. For each one-dimensional array $T[i][\cdot]$, there is a hash function $h_i : \{0, \ldots, n-1\} \rightarrow \{0, \ldots, K-1\}$, where n is the size of the key space. The hash functions are chosen uniformly at random to be pair-wise independent. We can view the data structure as an array of hash tables.

H	number of hash tables
K	number of counts per hash table
n	size of the key space
h_i	i^{th} hash function
$T[i][j]$	bucket j in hash table i
θ	threshold of heavy hitters
m	number of top hitters

Table 1: Sketch Notations

Update procedure: When an update (k_t, u_t) arrives, the update value u_t is added to the corresponding count $T[i][h_i(k_t)]$ in each hash table i.

Heavy hitter identification: Since the sketch data structure only records the values, not the keys, it is a challenge to identify the heavy-valued keys among all the keys hashed into the heavy buckets. In order to identify heavy hitters, we can keep a priority queue to record the top hitters with values above θ (as shown in [5]). An alternative is to perform intersections among buckets with heavy counts, which is proposed by Schweller *et al.* [14].

Point estimation: Let S be a sketch and X be a set of keys, which are known to be heavy hitters. The problem of *point estimation* is to estimate the total update value U_k for any key $k \in X$. This problem is the focus of our paper.

Count-Min: As proposed in [5], *countmin* is an existing method to reconstruct the value for any given key. The minimum value among all counts corresponding to the key is taken as an estimate of the value. Formally,

$$U_k^{\text{countmin}} = \min_{0 \leq i < H} T[i][h_i(k)]$$

is an estimate for the value U_k. Cormode and Muthukrishnan [5] proved that $U_k \leq U_k^{\text{countmin}}$ and that $U_k^{\text{countmin}} \leq U_k + \epsilon \|\mathbf{U}\|_1$ with probability δ, where $H = \lceil \frac{\mathbf{e}}{\epsilon} \rceil$, $K = \lceil \ln \frac{1}{\delta} \rceil$, and $\|\mathbf{U}\|_1 = \sum_{k=0}^{n-1} |U_k|$. In other words, *countmin* always overestimates with a certain error bound.

2.3 Related Work

Common applications of sketches include detecting heavy-hitters, finding quantiles, answering range/point queries and estimating flow size distribution [11].

Kumar *et al.* [9] used Expectation Maximization method to infer the flow size distribution from an array of counters, which can be viewed as a special case of sketch ($H = 1$).

Estan and Varghese [6] suggested an improved sampling method called *sample-and-hold*, with which flow amount is recorded only after individual entry for the flow is made. They also proposed *multi-stage filters* for data summary, which has the same data structure as sketch but uses a different update method called *conservative update*. When an update arrives, only the minimum valued bucket is incremented, whereas sketch increments counters of *all* corresponding buckets. The minimum counter of multi-stage filter can be used for point estimation, which is similar to the *countmin* approach.

Krishnamurthy *et al.* [8] proposed another point estimation method for sketch, which can be used in the Turnstile data stream model. The estimation U_k^{est} for a key k is given as $U_k^{\text{est}} = \text{median}\{U_k^i \mid 0 \le i < H\}$, where $U_k^i = \frac{T[i][h_i(k)] - SUM/K}{1-1/K}$ and $SUM = \sum_{j=0}^{K-1} T[0][j]$.

3 Our Approach

In this section, we explain the proposed *lsquare* method for point estimation. First, *lsquare* records the data flow information in a sketch. Then it constructs a linear system based on the sketch, and solves the system using linear least squares method. Below we first give a simple example and then formally describe the method.

3.1 A Simple Example

Suppose we have a data stream from 5 IP addresses. Let $U_0 = 5, U_1 = 4, U_2 = 3, U_3 = 9, U_4 = 16$ be the total amount of traffic for each IP. We record the flows into a sketch with $H = 2$ and $K = 3$, which has two hash functions $h_1(k) = k \bmod 3$ and $h_2(k) = (k \oplus 3) \bmod 3$, where $\oplus$ denotes bitwise-XOR. The sketch is given as:

	$j=0$	$j=1$	$j=2$
$T[0][j]$	$14^{0,3}$	$20^{1,4}$	3^2
$T[1][j]$	$14^{0,3}$	$19^{2,4}$	4^1

Here, $14^{0,3}$ means that U_0 and U_3 are hashed into the bucket, resulting in a count of 14. The goal is to reconstruct U_3 and U_4 from the sketch.

Solution using *countmin*: U_3^{countmin} = min$\{T[0][0], T[1][0]\}$ = 14 and U_4^{countmin} = min$\{T[0][1], T[1][1]\}$ = 19.

Solution using *lsquare*: First, we construct a linear system $A\mathbf{x} = \mathbf{b}$ with the constructed sketch. Vectors $\mathbf{x}$, $\mathbf{b}$ and matrix A are specified as follows.

$$\mathbf{x} = \begin{bmatrix} x_3 \\ x_4 \\ y \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 14 \\ 20 \\ 3 \\ 14 \\ 19 \\ 4 \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Here, x_3 and x_4 are variables for keys 3 and 4, and y is used to capture noise caused by keys that are not of our interest. Matrix A indicates which keys are hashed to which buckets, and vector $\mathbf{b}$ consists of values of all buckets. For example, we have the equation $x_3 + y = 14$ with bucket $T[0][0]$, which corresponds to the first rows of A and $\mathbf{b}$.

With the constructed linear system, we find the optimal solution of the linear system using linear least squares method: $\mathbf{x} = [10.5, 16.0, 3.5]^T$ (*i.e.*, $x_3 = 10.5$, $x_4 = 16.0$, $y = 3.5$). In this simple example, our method clearly produces much more accurate estimates than *countmin*.

3.2 Formal Description of *lsquare*

Let S be a sketch and $k_1, \ldots, k_m$ be the set of keys of our interest. Then we have an unknown variables vector $\mathbf{x} \in \mathbb{R}^{(m+1)\times 1} = [x_1, \ldots, x_m, y]^T$, where x_i is for the value of key k_i and y is an additional variable for noise caused by keys not in $\{k_i\}$, which is uniformly distributed over all buckets. We construct a matrix $A \in \{0,1\}^{HK\times(m+1)}$, showing which keys are hashed into which buckets, and a vector $\mathbf{b} \in \mathbb{R}^{HK\times 1}$, containing values of every buckets. The elements of A and $\mathbf{b}$ are specified as follows. For $i \in \{0, \ldots H-1\}$ and $j \in \{0, \ldots K-1\}$,

$$\begin{aligned} A_{Ki+j+1,\ell} &= \begin{cases} 1 & \text{if key } k_\ell \text{ is hashed into } T[i][j], \\ 1 & \text{if } \ell = m+1, \\ 0 & \text{otherwise,} \end{cases} \\ b_{Ki+j+1} &= T[i][j]. \end{aligned}$$

In general A is not a square matrix and may be rank deficient. In this case, a standard solution to $A\mathbf{x} = \mathbf{b}$ is the pseudoinverse solution $\mathbf{x} = A^+\mathbf{b}$, where A^+ is the pseudoinverse (*i.e.*, Moore-Penrose inverse [10, 13]) of matrix A. It is known that $\mathbf{x} = A^+\mathbf{b}$ provides the shortest length least squares solution to the system of linear equations $A\mathbf{x} = \mathbf{b}$. More precisely, it solves:

$$\text{minimize } \|\mathbf{x}\|_2^2 \quad \text{subject to } \|A\mathbf{x} - \mathbf{b}\|_2^2 \text{ is minimal,}$$

where $\|\cdot\|_2$ is the *Euclidean norm.*

Under the cash register data stream model, we can further improve the estimation accuracy by incorporating lower-bound and upper-bound constraints into the system. Specifically, we can use 0 as a lower bound for $\mathbf{x}$ and the *countmin* estimation as an upper bound. The pseudo-code for the resulting algorithm is given as follows.

```
vector lsquare(matrix A, vector b,
               vector countmin)
{
  x = pinv(A)*b;       // pseudoinverse
  x = max(x,0);        // non-negativity
  x = min(x,countmin); // upper bound: countmin
  return x;
}
```

Note that so far we use a single variable y to capture the effects of background noise. This assumes that we do not know any keys other than those of our direct interest. In case we do know extra keys, we can add them to $\{k_i\}$ and treat the corresponding x_i as additional noise variables. We will show in Section 4.3 that the use of additional noise variables significantly improves the accuracy of *lsquare*.

4 Evaluation

In this section we evaluate our *lsquare* method on two Internet trace data sets. Our results suggest that *lsquare* generally produces more accurate estimates than *countmin*. Even better accuracy can be achieved through the use of additional noise variables. In addition, the accuracy of *lsquare* degrades gracefully when less memory is available.

4.1 Data Sets

The Internet traffic data used in our evaluation is collected by National Laboratory for Applied Network Research (NLANR) [12]. We choose two sets of data: BELL-02 [2] and TERA-04 [15]. Brief information of the data sets is given in Table 2. Figure 1 shows the traffic amount of top 200 heavy hitters in two data sets. We can see that the traffic distributions are highly skewed.

	BELL-02	TERA-04
Time	2002/05/19 (1-2PM)	2004/02/09 (8-9AM)
Volume	8.371 GB	0.106 GB

Table 2: Data Set Information

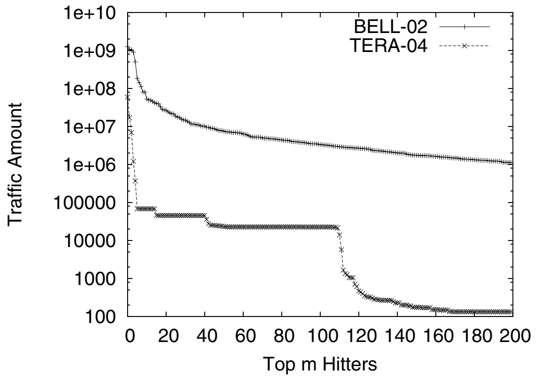


Figure 1: Traffic amount of top 200 hitters in BELL-02 and TERA-04

4.2 Error Metric

We use a relative error metric to evaluate the estimation. When evaluating an estimate for a specific key k, we use

$$E_k = \frac{U_k^{\text{est}} - U_k}{U_k}.$$

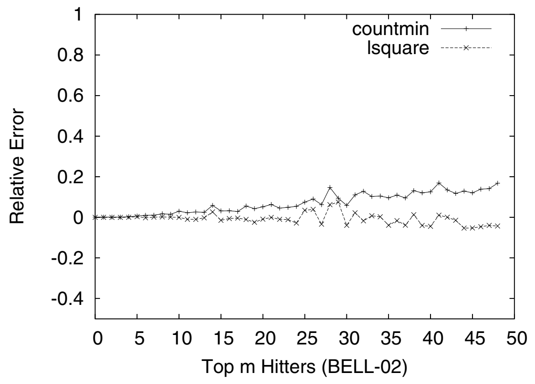
This metric gets close to 0 when the estimation is accurate and it can indicate whether we have overestimated or underestimated results. When we evaluate the estimation result as a whole, we use the average error

$$E = \sqrt{\frac{1}{|HH|} \sum_{k \in HH} \left(\frac{U_k^{\text{est}} - U_k}{U_k} \right)^2}$$

as a metric, where HH is the set of heavy hitters of our interest. The square of point error metric is used to avoid cancellation between positive and negative errors.

4.3 Accuracy

We first compare the accuracy of *lsquare* and *countmin* when a single variable y is used to capture the background noise (caused by keys not in HH). As a preliminary experiment, we calculate the estimation errors for top 50 heavy hitters using the two methods, with $H = 4$ and $K = 1024$



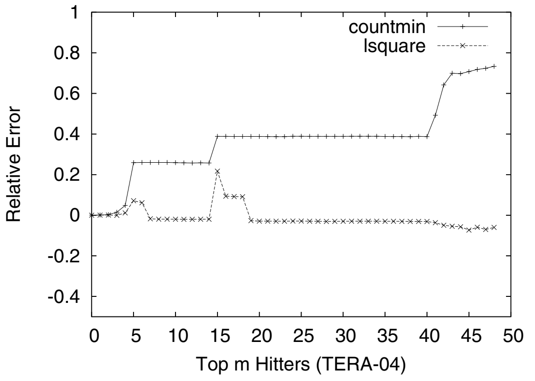


Figure 2: $K = 1024$, $H = 4$, $m = 50$: *lsquare* shows more accurate and stable estimation than *countmin*.

(Figure 2). We observe that the accuracy of *countmin* fluctuates depending on the data sets, whereas *lsquare* consistently gives more stable and accurate estimates.

We next demonstrate that better accuracy can be achieved when we use more variables to capture the noise effects. In Figure 3 we evaluate the accuracy of *lsquare* with a varying number of noise variables. For each data set, we calculate the estimation errors of top 20 heavy hitters in three cases. In the case of experiment "20-*lsquare*", just one noise variable y is used. Then top 21–50 hitters are considered as noise variables (in addition to y) in experiment "50-*lsquare*", and top 21–200 hitters in experiment "200-*lsquare*." As more noise variables are used, *lsquare* becomes more stable and accurate. In particular, *lsquare* has almost no errors in the case of "200-*lsquare*."

In addition, *lsquare* produces accurate estimates even for "light" hitters. In Figure 4, we calculate the estimation errors for top 200 hitters. In BELL-02 data set, *lsquare* shows relatively accurate estimation for top 160 hitters, where *countmin* is only good for top 40 hitters. We observe bigger accuracy difference between the two methods in TERA-04 data set: *lsquare* still has accurate estimation for top 170 hitters but *countmin* has good performance only for top 20 hitters. Moreover, the accuracy of *countmin* for light hitters is significantly lower.

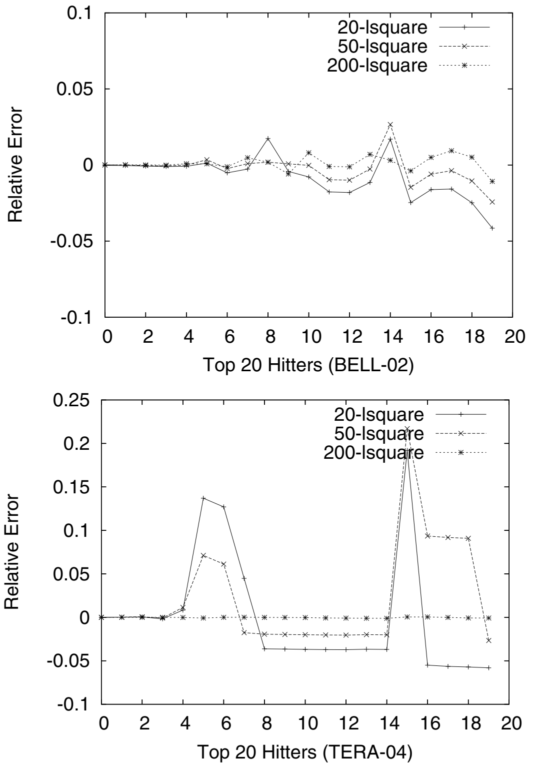


Figure 3: $K = 1024$, $H = 4$, $m = 20$: *lsquare* shows better accuracy when we use more noise variables.

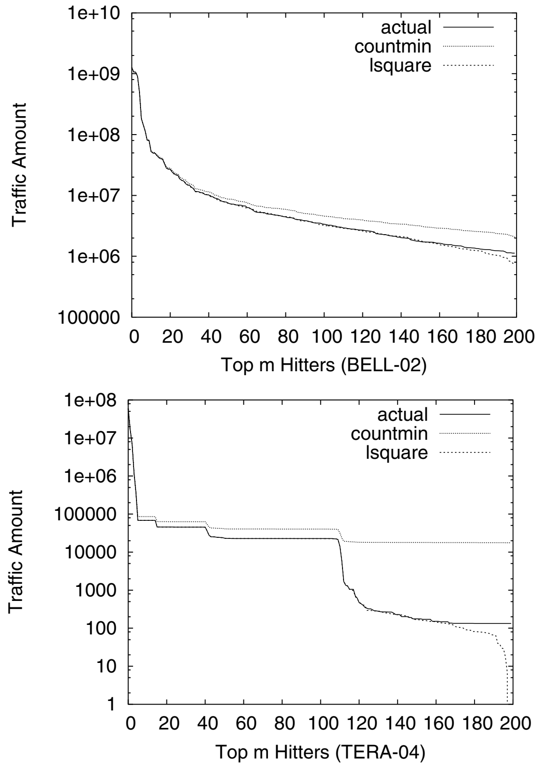


Figure 4: $K = 1024$, $H = 4$, $m = 200$: *lsquare* achieves good accuracy even for light hitters.

4.4 Tolerance with Limited Memory

We now evaluate the accuracy of *countmin* and *lsquare* under the constraint of limited memory. Since a sketch is usually located within an expensive memory SRAM for high-speed traffic monitoring, it is desirable to have accurate point estimates even if we reduce the size of the sketch.

First, we fix the number of buckets in a hash table K to be 1024 and vary the number of hash tables H. Next, we vary K with fixed $H = 4$. In Figure 5 and 6, we calculate the average error of the two methods for each sketch configuration. We can see clearly that the accuracy of *lsquare* degrades gracefully as the sketch gets smaller, whereas *countmin* gives inaccurate estimates in memory-limited situations.

To make the experiment more reliable regardless of the sketch configuration, we find the optimal combination for *countmin* in the given memory size after trying various combinations of H and K. Within the configuration where *countmin* shows the best accuracy, we evaluate the accuracy of *lsquare*. Once again, we observe better accuracy of the proposed method (Figure 7).

4.5 Time Performance

We have implemented our *lsquare* method in Matlab. The most time-consuming process in our method is solving the linear system $A\mathbf{x} = \mathbf{b}$. We make a preliminary evaluation regarding the time performance of our implementation us-

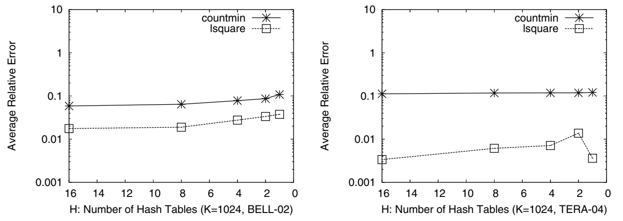


Figure 5: $K = 1024$, $H = 1, 2, 4, 8, 16$, $\theta = 0.1\%$: The number of hash tables has little impact on accuracy. *lsquare* consistently shows better accuracy than *countmin*.

ing a Pentium3-733MHz machine with 128 MBytes memory, operated by Linux Debian 3.0. In this experiment, we use a fixed sketch configuration (H=4, K=1024) and vary the number of heavy hitters we want to estimate. The results in Figure 8 show that the linear program solver can compute point estimations of 100 heavy hitters in about 2 seconds in the given configuration. We note that our current Matlab implementation has not been fully optimized and there is considerable room for further speedup. For example, we can replace the pseudoinverse function `pinv` with an iterative least-squares solver such as `lsqr` to take advantage of the sparsity of matrix A.

5 Conclusion and Future Work

In this paper, we propose a new approach for point estimation on sketches. Using extensive experiments with real In-

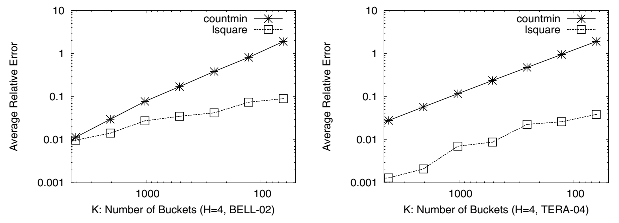


Figure 6: $K = 64, 128, 256, 512, 1k, 2k, 4k$, $H = 4$, $\theta = 0.1\%$: The number of buckets in a hash table has a big impact on accuracy. The accuracy of *lsquare* degrades more gracefully as the number of buckets decreases.

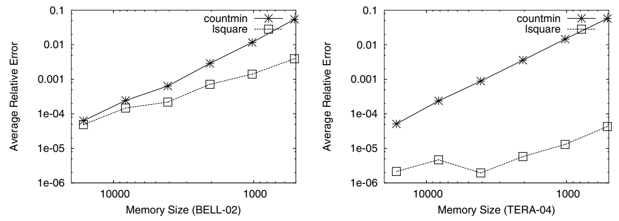


Figure 7: $H \times K = 512, 1k, 2k, 4k, 8k, 16k$, $\theta = 0.1\%$: For each memory size, we find the optimal sketch configuration for *countmin*. In that optimal configuration, we compare the accuracy of *lsquare* and *countmin*. In both data sets, *lsquare* shows better performance.

ternet data sets, we show that the proposed method *lsquare* is much more accurate than the best existing method *countmin*. *lsquare* achieves good reconstruction accuracy for both heavy and light hitters, at the expense of modest computation. Moreover, we have shown that the accuracy of *lsquare* degrades gracefully as memory decreases. To achieve accuracy comparable to *countmin*, *lsquare* in general requires much less memory.

This paper represents an early example on how traditional statistical inference techniques can be applied in the data stream context to infer characteristics of the input stream. Existing research on data stream computation so far has mainly focused on developing techniques that provide provable worst-case accuracy guarantees. Statistical inference techniques in contrast often pay more attention to properties like likelihood, unbiasedness, estimation variance etc. While these inference techniques may not provide any worst-case accuracy guarantees, they often perform very well on practical problems. In our future work, we plan to further explore how statistical inference techniques can be applied to data stream computation.

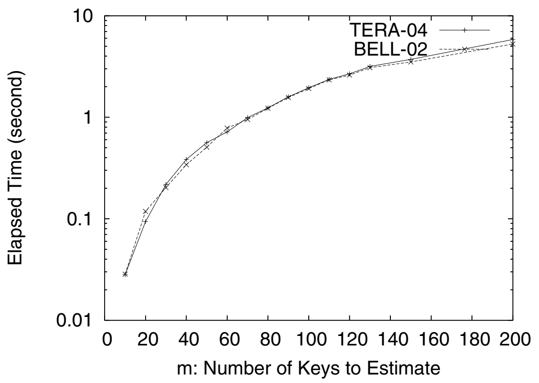


Figure 8: $K = 1024$, $H = 4$: This graph shows the elapsed time to execute the Linear System Solver with various numbers of heavy hitters.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Bell Labs I data set, 2002. `http://pma.nlanr.net/Traces/long/bell1.html`.

[3] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of ICALP '2002*, pages 693–703, 2002. `http://www.cs.princeton.edu/~moses/papers/frequent.ps`.

[4] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of ACM PODC '2003*, July 2003.

[5] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Proceedings of Latin American Theoretical Informatics (LATIN)*, pages 29–38, 2004.

[6] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *Proceedings of ACM SIGCOMM*, 2002.

[7] P. Gibbons and Y. Matias. Synopsis structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.

[8] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2003.

[9] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow distribution. *Proceedings of ACM Sigmetrics/Performance*, 2004.

[10] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395, 1920.

[11] S. Muthukrishnan. Data streams: Algorithms and applications, 2003. Manuscript based on invited talk from *14th SODA*. Available from `http://www.cs.rutgers.edu/~muthu/stream-1-1.ps`.

[12] NLANR. `http://www.nlanr.net/`.

[13] R. Penrose. A generalized inverse of matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:406–412, 1955.

[14] R. Schweller, A. Gupta, Y. Chen, and E. Parsons. Reversible sketches for efficient and accurate change detection over network data streams. *Proceedings of ACM SIGCOMM Internet Measurement Conference*, pages 207–212, 2004.

[15] Teragrid-I 10GigE trace, 2004. `http://pma.nlanr.net/Special/tera1.html`.

Understanding Congestion in IEEE 802.11b Wireless Networks

Amit P. Jardosh   Krishna N. Ramachandran   Kevin C. Almeroth   Elizabeth M. Belding-Royer
Department of Computer Science
University of California, Santa Barbara
$\{$*amitj,krishna,almeroth,ebelding*$\}$*@cs.ucsb.edu*

Abstract

The growing popularity of wireless networks has led to cases of heavy utilization and congestion. In heavily utilized wireless networks, the wireless portion of the network is a major performance bottleneck. Understanding the behavior of the wireless portion of such networks is critical to ensure their robust operation. This understanding can also help optimize network performance. In this paper, we use link layer information collected from an operational, large-scale, and heavily utilized IEEE 802.11b wireless network deployed at the 62nd Internet Engineering Task Force (IETF) meeting to study congestion in wireless networks. We motivate the use of *channel busy-time* as a direct measure of channel utilization and show how channel utilization along with network throughput and goodput can be used to define *highly congested*, *moderately congested*, and *uncongested* network states. Our study correlates network congestion and its effect on link-layer performance. Based on these correlations we find that (1) current rate adaptation implementations make scarce use of the 2 Mbps and 5.5 Mbps data rates, (2) the use of Request-to-Send/Clear-to-Send (RTS–CTS) prevents nodes from gaining fair access to a heavily congested channel, and (3) the use of rate adaptation, as a response to congestion, is detrimental to network performance.

1 Introduction

The occurrence of a high density of nodes within a single collision domain of an IEEE 802.11 wireless network can result in congestion, thereby causing a significant performance bottleneck. Effects of congestion include drastic drops in network throughput, unacceptable packet delays, and session disruptions. In contrast, the *back-haul wireline* portion of a wireless network is typically well provisioned to handle the network load. Therefore, there arises a compelling need to understand the behavior of the wireless portion of heavily utilized and congested wireless networks.

To fulfill our endeavor of studying the performance of congested wireless networks, we collected link-layer traces from a large-scale wireless network at the 62nd Internet Engineering Task Force (IETF)[1] meeting held in Minneapolis, Minnesota. The meeting was held March 6–11, 2005 and was attended by 1138 participants. Almost all of the participants used laptops or other wireless devices. The wireless network consisted of 38 IEEE 802.11b access points deployed on three adjacent floors of the venue. Our traces collected over two days at the meeting consist of approximately 57 million frames and amount to 45 gigabytes of data. An analysis of the data shows that the large number of participants and access points resulted in heavy utilization of the wireless network with multiple periods of congestion.

Through the analysis of the IETF network, we aim to understand the impact of congestion in wireless networks by answering questions such as: (1) what does congestion mean in a wireless network? (2) how do we identify it? (3) what are the challenges in the congestion analysis? (4) what are the effects of congestion on link-layer properties? Answering these questions is non-trivial. The difficulty arises because of several reasons. First, the IEEE standards do not specify a number of often used protocol features. Examples include the rate adaptation protocol and the transmission power control scheme. Implementations of such features are vendor-specific and their details are often proprietary. Second, monitoring tools for sniffing link-layer information are limited in their capabilities because they cannot capture *all* relevant information about *all* transmitted packets due to either fundamental hardware limitations, the proprietary nature of hardware and software, or hidden terminals. Finally, we performed the monitoring in an uncontrolled environment. This precluded any parameterized behavioral analysis which might be otherwise possible in a controlled laboratory setting.

In our effort to answer the above questions, we first show how channel utilization can be used to identify various states of congestion in the wireless medium. Further, we use channel utilization to explain the behavior of the MAC layer. The behavior is analyzed by studying factors such as the channel busy-time, effectiveness of the Request-to-Send/Clear-to-Send (RTS–CTS) mechanism, frame transmission and reception, and acceptance delays. Based on our analysis, we make the following main observations:

- The use of RTS–CTS by a few nodes in a heavily congested environment prevents those nodes from gaining fair access to the channel.
- The number of frame transmissions at 1 Mbps and 11 Mbps are high for all congestion levels. Current rate-adaptation implementations make scarce use of the 2 Mbps and 5.5 Mbps data rates irrespective of the level of congestion.
- At high congestion levels, the time to successfully transmit a large frame sent at 11 Mbps is *lower* than for a small frame sent at 1 Mbps.
- At high congestion levels, the time consumed by frames transmitted at 11 Mbps is only about half the time consumed by frames transmitted at 1 Mbps. Yet the number of bytes transmitted at 11 Mbps is approximately 300% more than at 1 Mbps.

These observations offer important insight into the operation and performance of congested wireless networks. We believe that these observations hint at significant deficiencies in the 802.11b protocol and its implementations. Even though the above observations are specific to the IETF network, we believe they will generally be applicable in other network configurations because of the large diversity in wireless hardware and network usage patterns recorded in our data set.

We believe this paper is the first of its kind to empirically analyze a heavily-congested wireless network. While we are unable to totally understand all the observed network behavior because of the above noted challenges, we believe that we offer significant insight into the behavior of a heavily congested network. We hope that the insight can be utilized to design better systems and protocols. We also hope that analysis of congestion in wireless networks will be the focus of future monitoring efforts.

The remainder of this paper is organized as follows: Section 2 motivates the importance of understanding congestion in wireless networks. An overview of the IEEE 802.11b MAC protocol is given in Section 3. Section 4 describes the data collection methodology and the challenges of vicinity sniffing in large-scale networks. Section 5 describes a method for measuring network congestion. The effects of congestion on data packet retransmissions, frame sizes, and data rates are discussed in Section 6. Section 7 presents the conclusions from our study.

2 Related Work and Motivation

A large number of studies have analyzed the performance of wireless networks. We summarize a representative sample of the existing work below.

Several studies have utilized measurements from production wireless networks to compute traffic models [2, 7, 13, 14, 19] and mobility models [3, 6, 22]. The primary focus of these studies has been to either investigate transport and application layer performance through the analysis of traffic captured on the wireline portion of the network, or utilize SNMP and syslog information from access points to model mobility and association patterns. Few studies have analyzed the performance of the wireless portion of deployed networks. Yeo et al. capture link-layer information to analyze the performance of a small-scale campus wireless network [23]. Mishra et al. use a sniffer to study the AP hand-off performance in a controlled experiment [15].

The effect of congestion on the performance of the various protocol layers has been studied extensively using either simulations or analytical methods. Cen et al. propose algorithms for distinguishing congestion from wireless network losses [5]. The algorithms provide a basis for optimizing TCP parameters such as back-off intervals and congestion window sizes. Several methods for the optimization of the 802.11 protocol in congested environments have been suggested [16, 20, 21]. Techniques have been proposed that optimize 802.11 protocol performance by either adjusting frame sizes in high bit-rate environments [16, 21] or varying the protocol contention window [1]. Heusse et al. analyze problems with multi-rate adaptation in the 802.11b protocol [8]. They suggest that because frames transmitted at low data rates occupy more time in the channel compared to frames transmitted at high data rates, hosts utilizing the high data rates suffer a penalty. This penalization is considered to be an anomaly in the 802.11 Distributed Coordination Function (DCF) Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) protocol. Finally, Cantieni et al. theoretically evaluate the effect of congested wireless networks on frames transmitted at different rates [4]. We experimentally confirm their analysis in Section 6.

The above studies do not offer an experimental evaluation of link-layer performance in heavily utilized and congested wireless networks. We believe that gaining a deep understanding of the real-world performance of the link-layer in congested networks is important. The insight can help in the design of robust protocols and implementations to handle congestion related problems more efficiently.

Rodrig et al. analyze the performance of a wireless network deployed at a conference by sniffing traffic from the wireless medium in the vicinity of a set of access points [17]. They discuss several challenges with wireless network sniffing and analyze current rate adaptation implementations and the overhead associated with IEEE 802.11b MAC transmissions.

Our initial efforts to understand the performance of heavily congested wireless networks is described in a previous paper [10]. Our work therein proposes a reliability metric that utilizes the reception of beacon frames from access points to compute link reliability. Our preliminary finding is that link reliability can be used to *estimate* congestion and explain its effects. On the other hand, the thesis of this paper is to propose a metric to determine congestion levels and to provide insight into the performance of the link-layer based on the congestion levels.

3 IEEE 802.11b DCF Protocol: Overview

This section summarizes the operation of the IEEE 802.11 Distributed Coordination Function (DCF) protocol. We limit the scope of the protocol description to aspects that are essential for a better understanding of the operation and functions discussed in this paper.

The IEEE 802.11b DCF protocol is designed to manage and reduce contention in the wireless communication medium in a *fair* manner. The algorithm used by the protocol is known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). According to the algorithm, when a node wants to transmit a frame, the station is required to first sense if the communication medium is busy. If it is, the station waits for a specific period of time known as the *Backoff Interval* (BO) and then tries to sense the medium again. If the channel is not busy, the station transmits the frame to the intended destination. The destination sends an acknowledgment message to the source of the frame upon successful reception of the frame. If the source does not receive an acknowledgment within a specific period of time, it tries to re-send the frame. Broadcast messages do not require the destination to send an acknowledgment of reception.

Contention in the communication medium can be further reduced using Request-To-Send (RTS) and Clear-To-Send (CTS) messages between sender-receiver pairs. A sender transmits an RTS with information about the size of the data frame to come and the channel time to be consumed by the data frame. If the receiver is free to receive the data frame, it sends a CTS back to the sender. At the same instant, other stations in the vicinity of the sender-receiver pair record the estimated time for data transmission and backoff until the channel becomes free again. The RTS–CTS mechanism is a technique for alleviating collisions caused because of the hidden terminal problem.

Timing sequence: Figure 1 shows the timing and sequence of frames and delays used by the 802.11 protocol [11]. The delays that precede and follow the transmission of control frames (RTS, CTS or ACK) or data frames are called Inter-Frame Spacings (IFS). Before the transmission of an RTS, stations are required to wait for a time equal to the Distributed IFS (DIFS). On the other hand, a destination station is required to send a CTS or an ACK frame within a Short IFS (SIFS) amount of time after the reception of RTS and DATA frames from the source, respectively. If the sending station senses the medium busy during the DIFS interval, it chooses a BO from a range and then an associated timer starts counting down to zero. If the channel is sensed busy during the BO, the sending station stops the countdown and resumes the countdown when the channel is idle again. When the timer counts down to zero, the sending station attempts to transmit the frame if the channel is idle for a period of DIFS. If the channel is busy, another BO is chosen from an exponentially increased range and the process repeats.

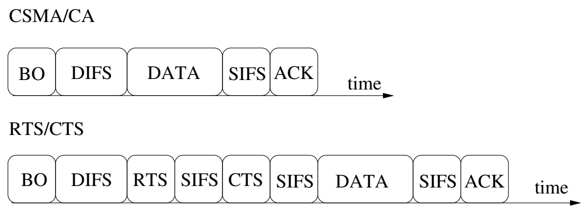


Figure 1: Frame and delay sequence diagram for CSMA/CA and RTS–CTS.

Multirate adaptation: To increase the probability of successful delivery of frames, wireless card vendors typically utilize a *multirate adaptation* algorithm that dynamically adapts the rates at which frames are transmitted. The rationale is that, at low rates, frames are more resilient to bit errors and hence are likely to be successfully received. The disadvantage, however, is that low data rates result in poor throughput performance. The IEEE 802.11 standards do not specify a rate adaptation scheme. As a result, 802.11 chipset manufacturers and driver developers can implement any suitable rate adaptation scheme. A popular technique is based on the auto rate fallback (ARF) scheme [12]. A typical ARF implementation reduces the transmission rate whenever packet drops occur and increases the rate upon successful delivery of a train of packets.

4 Data Collection Methodology

This section describes the IETF wireless network architecture, our monitoring framework, and a set of monitoring challenges for heavily utilized wireless networks.

4.1 The IETF Wireless Network

The IETF wireless network was comprised of 38 Airespace[2] 1250 Access Points (APs) distributed on three adjacent floors. Each AP supported both the IEEE 802.11a and IEEE 802.11b protocol standards; however, in our experiment we only analyze the operation of the IEEE 802.11b-based wireless network. Each physical AP supported four *virtual* or *logical* APs. Thus, a total of 152 virtual APs (38 physical APs $\times 4$ per physical AP) were available at the conference location. In this paper we use the term *AP* for a *virtual AP*. Figures 2 and 3 show the placement of APs in the rooms where we conducted our measurement and collection activities during the day and late evening sessions, respectively. There were 23 physical APs placed on one floor of the conference venue. The other 15 physical APs were located on the two adjacent floors. For the late evening sessions, the temporary walls between ballrooms D, E, F, and G were removed to form a single large ballroom.

In order to optimize network performance, the Airespace APs are designed to support dynamic channel assignment, client load balancing, and transmission power control. Dynamic channel assignment refers to the technique that switches the AP's operating channel, depending on parameters such as traffic load and the number of users associated with the AP. Client load balancing refers to the technique that controls per-AP user associations. The transmission power control regulates the power at which an AP transmits a frame. Unfortunately, technical details about these three optimizations are proprietary. Nevertheless, we observed that wireless network traffic was fairly well distributed between the three orthogonal channels 1, 6, and 11. Also, the access points were observed to switch channels dynamically to balance the number of users and traffic volume on the three channels.

4.2 The Data Collection Framework

The method we used to collect data from the MAC layer is called *vicinity sniffing* [10]. Our vicinity sniffing framework consisted of three *sniffers*, IBM R32 Think Pad laptops. Each sniffer was equipped with a Netgate 2511 PCMCIA 802.11b radio. The radios were configured to capture packets in a special operating mode called the *RFMon* mode. The RFMon mode enables the capture of regular data frames as well as IEEE 802.11b management frames. In addition, the RFMon mode records information for each captured packet. This information includes the send rate, the channel used for packet transmission, and the signal-to-noise ratio (SNR) of the received packet. Because the Airespace access points were expected to switch between the 802.11b channels 1, 6, and 11, each sniffer was configured to sniff on one of the three different channels for the duration of each session. The packets were captured using the sniffer utility *tethereal*. The snap-length of the captured packets was set to 250 bytes in order to capture only the RFMon, MAC, IP and TCP/UDP headers.

The data capturing process was conducted using two different placement configurations, one during the day and the second during the late evening sessions. The late evening sessions are called *plenary sessions*.

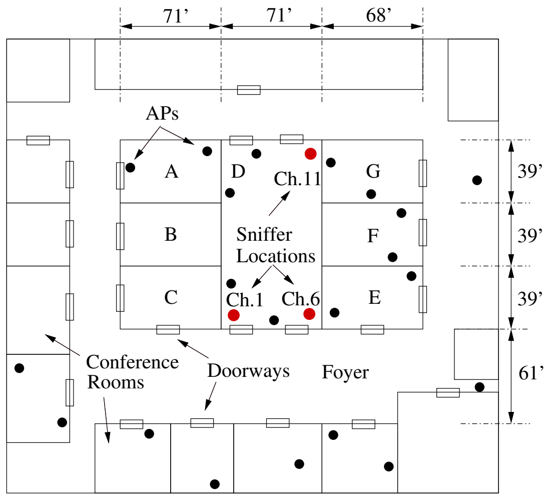


Figure 2: Sniffer locations during the day session.

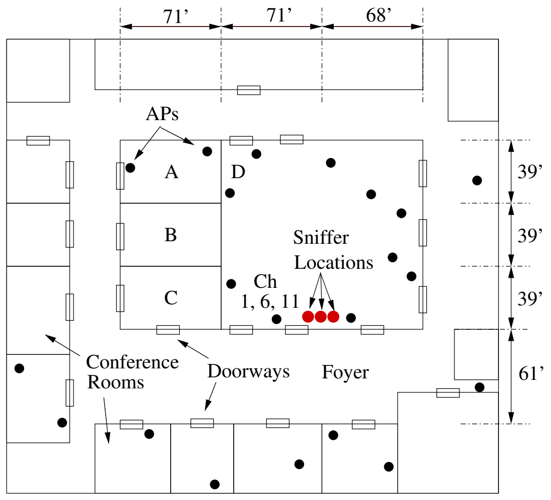


Figure 3: Sniffer locations during the late evening session.

Day sessions: The day sessions were held between 09:00 hrs and 17:30 hrs on March 6–11, 2005. The day sessions were split into 6 to 8 parallel tracks and each track was held in one of the several meeting rooms shown in Figure 2. The parallel session tracks were held at three intervals during the day: 09:00 hrs to 11:30 hrs, 13:00 hrs to 15:00 hrs, and 15:30 hrs to 17:30 hrs. We chose to place the three sniffers in one of the busiest and largest meeting rooms. The placement of the three sniffers is shown in Figure 2. Data was collected during the day sessions held on March 9, 2005.

Plenary sessions: Plenary sessions were held in a single large meeting room where all the IETF members congregate to discuss administrative issues. The plenary sessions at the 62nd IETF were held between 19:30 hrs and 22:30 hrs on March 9 and 10, 2005. Data was collected during the second plenary session held on March 10, 2005. Figure 3 shows the placement of the access points during the plenary session and the single point at which the three sniffers were co-located.

4.3 Data Sets

Wireless network data collected from the IETF network was separated into two sets named the *day session* and the *plenary session*. Table 1 shows the day, time, and channel that each of the two sets represents. Our collection framework recorded a total of 28.6 million data frames, 27.05 million acknowledgment frames, 40,000 RTS frames, and 17,490 CTS frames during the day and the plenary sessions cumulatively. The use of the RTS–CTS mechanism is generally turned off by default on wireless devices and its use is optional. The data indicates that the use of the RTS–CTS mechanism for channel access by conference participants was minimal.

Per-AP traffic: Figure 4(a) shows the number of data and control frames sent and received by the 15 most active APs out of 152 APs for the day and plenary sessions in decreasing order. We observe that the 15 most active APs during the day session sent and received 90.33% of the total 40.81 million frames, and the 15 most active APs during the plenary session sent and received 95.37% of the total 16.81 million frames.

Number of users: Figure 4(b) shows the instantaneous number of users associated with these access points for the two data sets. For visual clarity, each point on the graph represents the mean number of users in a 30-second interval. We observe that at 15:48 hrs during the day session, a maximum of 523 users were associated with the network, while at 20:45 hrs during the plenary session, a maximum of 325 users were associated with the network. This graph shows that the network was used by a large number of users for almost all of the collection period. In comparison with previous wireless network performance studies, we believe that the number of user associations and the number of frames sent and received by the APs during the day and plenary sessions confirms that the network is large, heavily utilized, and unique in its scale and usage. These traits make the evaluation of the information collected from the network critical and necessary for a clear understanding of heavily utilized and congested networks.

Data set	Day	Ch	Time
Day	March 9 2005	1	11:53–17:30 hrs
	March 9 2005	6	11:54–17:30 hrs
	March 9 2005	11	11:56–17:30 hrs
Plenary	March 10 2005	1	19:30–22:30 hrs
	March 10 2005	6	19:31–22:30 hrs
	March 10 2005	11	19:32–22:30 hrs

Table 1: The two sets of IETF wireless network data.

4.4 Vicinity Sniffing Challenges

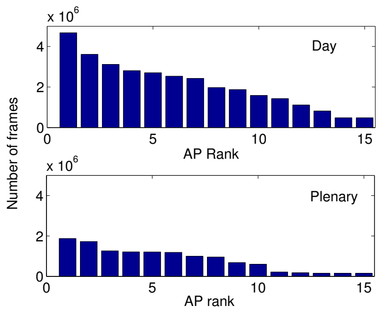
Vicinity sniffing is a technique used to capture control, management, and data frames transmitted by user devices and APs on the wireless portion of the network. In our experiment, we conducted vicinity sniffing using two different sets of sniffer locations for the day and plenary sessions. Vicinity sniffing is the best currently available method to collect link layer information from an operational wireless network. However, the utility of vicinity sniffing is limited by the following factors:

Choice of sniffer locations: The location of one or more sniffers affects the quantity and quality of frames that can be captured from the network. With *a priori* information about the AP topology and the expected number of frames transmitted to and from the APs, sniffers can be strategically and conveniently placed in the vicinity of those APs. At the IETF, this information was obtained by studying meeting schedules, gathering attendee statistics from meeting organizers, and conducting preliminary activity tests on March 8, 2005. The tests involved the placement of sniffers in different meeting rooms for a short period of time to capture a snapshot of the activity of the APs in the room. These tests allowed us to estimate the behavior of network traffic, number of users, per-AP traffic, and per-channel traffic.

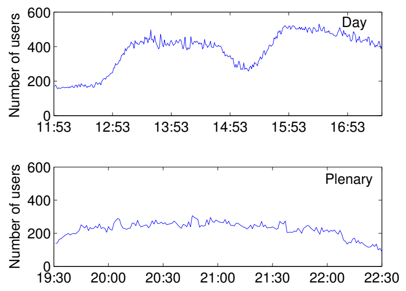
With the information obtained from the preliminary tests and the assumption that users of the wireless network were spread out in different conference rooms, we decided to place three sniffers in three different locations of a single room for the day sessions (shown in Figure 2). This placement allowed us to capture critical data sets from APs and user devices in and around the room. Due to logistical limitations, we co-located the sniffers at a single point during the plenary session, such that a majority of the traffic transmitted by the users and APs in the room could be captured. Since most attendees congregated in the same large room, we assumed that the placement of sniffers at a location *in* the room would enable us to capture a large portion of the relevant network traffic. The placement of the sniffers during the plenary session is shown in Figure 3.

Unrecorded frames: One of the critical challenges of vicinity sniffing is that the sniffers cannot record all frames that are transmitted over the communication channel. An unrecorded frame belongs to one of three different categories: (1) frames dropped due to bit errors in received frames, (2) hardware limitations that cause dropping of frames during high load conditions [23], and (3) frames that could not be recorded because of the hidden terminal problem.

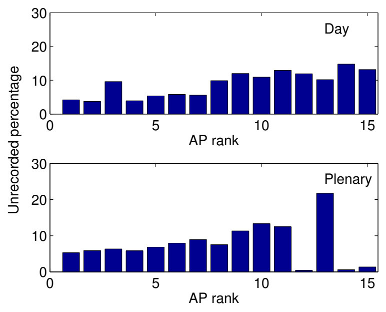
If the number of unrecorded frames is large, the conclusions drawn from the data set could be inaccurate. Therefore, in this section we discuss the techniques we use to estimate the number of unrecorded control and data frames and the impact of not capturing these frames. The number



(a) Total number of data and control frames sent and received by the 15 most active APs. Data frames for each AP are represented for the day and plenary sessions.



(b) Number of users associated with the wireless network during the day and plenary sessions, averaged over 30 second intervals.



(c) The unrecorded frames percentage computed for the data and control frames sent and received by the 15 most active APs.

Figure 4: Data frames, control frames, and number of users statistics.

of unrecorded DATA, RTS, and CTS frames can be estimated using the following techniques.

Data frames: To estimate the number of data frames that were unrecorded by the sniffers, we leverage the DATA–ACK frame arrival atomicity of the IEEE 802.11b DCF standard. The atomicity policy states that if a DATA frame is successfully received by a device in a network, the receiving device should send an ACK after a SIFS delay. No other device in the reception range is allowed to transmit frames during this interval. In other words, when an ACK frame occurs in our traffic logs, we expect a DATA frame to precede it. The source of the DATA frame must be the receiver of the ACK. If this DATA frame is missing, we can assume that our sniffers were unable to capture it.

RTS frames: The use of the RTS–CTS exchange of messages between a source and a destination is optional. However, in our data sets, we detected a limited use of RTS and CTS frames. Consequently, we were able to leverage the RTS–CTS frame arrival atomicity of the IEEE 802.11b DCF standard. It states that if an RTS is successfully received, the receiver may send a CTS after an SIFS delay. No other device in the reception range of the sending device is allowed to transmit frames during this interval. In other words, if a CTS frame is encountered in our data set, we expect an RTS frame to precede it and the receiver of the RTS must be the source of the CTS. If this RTS frame is missing from the data set but the CTS frame was recorded, we can determine that the sniffers were unable to record the RTS.

CTS frames: The number of uncaptured CTS frames can be derived by utilizing the RTS–CTS–DATA frame arrival atomicity of the IEEE 802.11b DCF standard. It states that if RTS and DATA frames are recorded, the receiver of the RTS must have sent a CTS frame following the RTS frame by an SIFS delay. The standard states that the station that sends the RTS will send the following DATA frame only when it receives a CTS from the destination station.

Unfortunately, a drawback of these techniques is that, if both the DATA and ACK frames are missing, or both the RTS and CTS frames are missing, or all three RTS, CTS, and DATA frames are missing, the techniques will fail to determine the DATA, RTS, and CTS frames, respectively, as unrecorded. However, the techniques do provide a close enough estimate of the number of unrecorded frames.

The *unrecorded percentage* is defined as the percentage of the total frames (control and data frames) that were detected as unrecorded over the sum of the total frames recorded and unrecorded in our data set. We compute the unrecorded percentage using Equation 1 as follows:

$$Unrecorded\_\% = \frac{unrec\_frames}{unrec\_frames + captured\_frames} \quad (1)$$

The *unrecorded percentage* for each of the 15 most active APs during the day and the plenary sessions is shown in Figure 4(c). The list of APs are ranked in decreasing order of the number of frames sent and received by the APs as shown in Figure 4(a). The figure shows that the unrecorded percentage for the first 15 APs varies between 3% to 15% during the day session, and between 5% to 20% during the plenary session. Based on these values, we assume that the results obtained from the data sets would not be significantly altered even if the occurrence of these frames could be accurately determined. However, to improve the accuracy of the results, we believe that future experiments should use a greater number of sniffers and better hardware to reduce the number of unrecorded frames caused by hidden terminals and hardware limitations.

5 Defining Congestion

On the Internet, a *network link* is said to be congested when the *offered load* on the link reaches a value close to the capacity of the link. In other words, we can define congestion as the state in which a network link is close to being completely utilized by the transmission of bytes. In a similar manner, wireless network congestion can be defined as the state in which the transmission channel is close to being completely utilized. The extent of utilization can be measured using a *channel busy-time* metric given as the fraction of a set period of time that a channel is busy.

In this section, we show how channel utilization is used in conjunction with the computed throughput and goodput of the channel to estimate a set of utilization thresholds to identify *levels of congestion* in the network. We also show that the effects of congestion on link layer behavior can be better explained by defining *levels* of congestion as compared to *exact values* of utilization.

5.1 Channel Utilization

Channel utilization for a set period of time is computed on a percentage scale. In our study, we choose to use one second as the period. We find that this interval is an appropriate granularity in our analysis.

The utilization of a network channel per second is computed by adding (1) the time utilized by the transmission of *all* data, management, and control frames recorded by our sniffers, and (2) the total number of delay components such as the Distributed Inter-frame Spacing (DIFS) and Short Inter-frame Spacing (SIFS) during the same second. These delays form a part of the channel utilization computation because, during this period, the medium remains *unshared* between the stations in the network. The communication channel is *unshared* when no other station in the vicinity of the station that holds the channel can transmit frames for the specified delay time. In this paper we use delay component values suggested by Jun et al. [11]. Table 2 shows the delay in microseconds for delay components of the IEEE 802.11b protocol.

As previously described, Figure 1 shows the timing diagram for the CSMA/CA and (RTS/CTS) mechanisms. The diagram suggests a specific ordering of delay components. For instance, a DATA packet is preceded by SIFS delays, an RTS packet is preceded by a DIFS delay, and an ACK packet is preceded by an SIFS delay. In the heavily utilized IETF network where hundreds of users are associated with the network simultaneously, at any given instant, a minimum of a single user is *ready* to send a packet. In other words, we assume that at least one station has a BO timer equal to zero, at any instant. Therefore, the average time spent in the Back-off (BO) state will be equal to zero, i.e., $D_{BO} = 0$.

The delay components specified in Table 2 suggest that, first, the channel utilization increases for larger data frames since a larger number of bytes take greater time to transmit. Second, channel utilization increases with a decrease in the rate at which data frames are transmitted. And third, the data frame Physical Layer Convergence Protocol (PLCP) header is always transmitted with a fixed delay equal to D_{PLCP}.

Delay Component	Delay (μsec.)
D_{DIFS}	50
D_{SIFS}	10
D_{RTS}	352
D_{CTS}	304
D_{ACK}	304
D_{BEACON}	304
D_{BO}	0
D_{PLCP}	192
$D_{DATA}(size, rate)$	$D_{PLCP} + 8 \times (\frac{34+size}{rate})$

Table 2: Delay components specified in microseconds.

To accurately compute the channel utilization for a packet encountered in our data set, we use the timing diagram shown in Figure 1 and the delay component values in Table 2. Depending on the type of control frame and the rate and size of a data frame, the channel busy-time for the frame is computed as follows:

Data frames: A DIFS interval occurs before each data frame, either immediately before the data frame, in the case when the RTS–CTS mechanism is not utilized, or before the RTS frame, in the case when RTS–CTS is used. The DIFS delay interval is used for the busy-time computation of a data frame. The channel busy-time (CBT) for a data frame of size, *S* bytes, sent at a rate *R*, is computed using Equation 2.

$$CBT_{DATA} = D_{DIFS} + D_{DATA}(S, R) \tag{2}$$

RTS frames: In the case when RTS frames are encountered in our data set, the CBT for the frames is computed using Equation 3.

$$CBT_{RTS} = D_{RTS} \tag{3}$$

CTS frames: When a CTS frame is encountered in our data set, Figure 1 suggests that the CTS frame is transmitted following an SIFS delay after the RTS frame was received. Hence, the CBT for CTS frames is computed using Equation 4.

$$CBT_{RTS} = D_{SIFS} + D_{CTS} \tag{4}$$

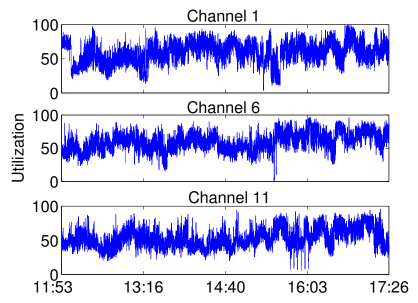
ACK frames: When an ACK frame is encountered in our data set, Figure 1 suggests that the ACK frame is transmitted following an SIFS delay after the preceding data frame was received. Hence, the CBT for ACK frames is computed using Equation 5.

$$CBT_{ACK} = D_{SIFS} + D_{ACK} \tag{5}$$

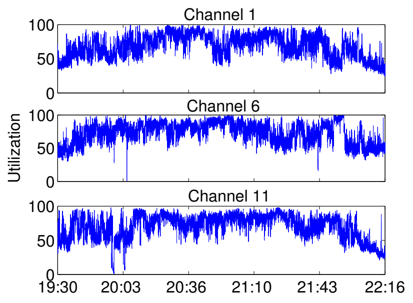
Beacon frames: A beacon frame is typically sent by each AP in the network at 100 millisecond intervals. The beacon frames are preceded by a DIFS delay interval. Hence, when a beacon frame is encountered in the data set, the CBT is computed using Equation 6.

$$CBT_{BEACON} = D_{DIFS} + D_{BEACON} \tag{6}$$

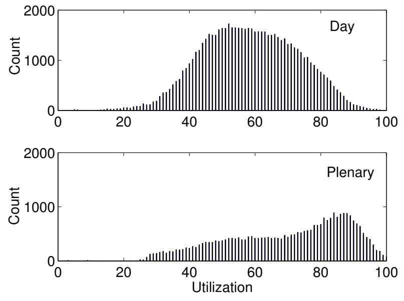
The CBT of the channel for a one second interval, *t*, is the total delay computed for all data and control frames



(a) Percentage utilization for the day session.



(b) Percentage utilization for the plenary session.



(c) Frequency of percentage utilization values.

Figure 5: Utilization percentage and frequency distribution.

that are transmitted within a second. Therefore, if *r(t)* RTS frames, *c(t)* CTS frames, *a(t)* ACK frames, *b(t)* beacon frames, and *d(t)* data frames are encountered during the same one second interval, the total CBT for the interval is calculated using Equation 7.

$$\begin{aligned} & CBT_{TOTAL}(t) \\ & = (r(t) \times CBT_{RTS}) + (c(t) \times CBT_{CTS}) \\ & + (a(t) \times CBT_{ACK}) + (b(t) \times CBT_{BEACON}) \\ & + (\sum_{i=1}^{d(t)} CBT_{DATA}(S_i, R_i)) \end{aligned} \tag{7}$$

The percentage channel utilization over the one second interval, $U(t)$, is computed using Equation 8 as follows:

$$U(t) = \frac{CBT_{TOTAL}(t)}{10^6} \times 100 \tag{8}$$

Equation 8 is used to compute the percentage utilization of the channel per second during the day and the plenary sessions.

Utilization frequency: The utilization computations are graphed on a time-series plot in Figure 5(a) for the day session and Figure 5(b) for the plenary session. Figure 5(c) is a histogram that shows the frequency of percentage utilization for the day and plenary sessions; for instance, the channel was 53% utilized for 1823 seconds during the day session. The histograms indicate that during the day the channel most often experienced about 55% utilization, while during the plenary, the channel was most often utilized at about 86%. During the day session, users were distributed across all the meeting rooms, and hence, fewer data and control frame transmissions were collected by the sniffers. On the other hand, users congregated closer to the sniffers during the plenary session, and hence, a larger number of transmissions can be collected by the sniffers. The proximity of users to the sniffers thus results in a higher channel utilization levels.

Figure 5(c) shows that there is not a significant period of time when the network was 0–30% or 99–100% utilized and so it is difficult to use our data set to characterize the behavior of the network. Thus, the evaluation of link-layer behavior in this paper focuses on the periods when the network was utilized between 30–99%.

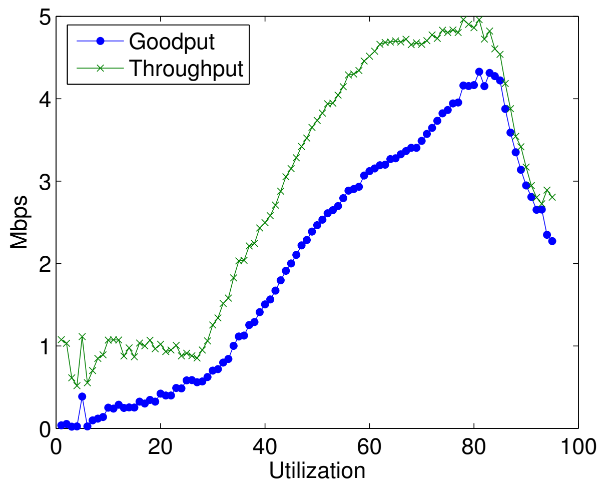


Figure 6: Channel throughput and goodput per second calculated at the corresponding channel utilization.

5.2 Throughput and Goodput

The *throughput* of the channel for a one second interval is the sum of the total number of bits of all recorded frames transmitted over the wireless channel during a one second interval. The goodput of the channel is the total number of recorded bits of all the control and *successfully acknowledged* data frames transmitted over the wireless channel during a one second interval.

Figure 6 shows the throughput and goodput of the channel versus channel utilization. Each point value *y* in the figure represents the average throughput or goodput over all one second intervals during the day and plenary sessions that are *x*% utilized. The number of one second intervals is equal to the frequency of percentage utilization of the channel shown in Figure 5(c). Figure 6 indicates that as the channel utilization increases from 30% to 84%, the average throughput of the channel increases to 4.9 Mbps and the average goodput increases to 4.4 Mbps. The average throughput at 84% channel utilization is closest to the achievable theoretical maximum throughput [11]. Since the computation of throughput includes all the transmitted frames, the calculated throughput value for each utilization percentage is higher than the corresponding value of goodput.

For channel utilization between 84% and 98%, we observe a significant decrease in throughput, from 4.9 Mbps to 2.8 Mbps, and a decrease in goodput, from 4.4 Mbps to 2.6 Mbps. This decrease in the throughput and goodput with a concomitant increase in channel utilization of the wireless network is due to the multirate adaptation algorithms. As channel utilization increases, a large number of frame errors and retransmissions occur. As retransmissions increase, most network cards decrease the rate at which each data packet is transmitted. At lower data rates (i.e., 1 Mbps), frames occupy the channel for a longer period of time, and hence, fewer bytes are transmitted over the channel. Heusse et al. state that the use of lower data rates for data frames penalizes the delivery of data frames transmitted at higher rates and is an anomaly of the IEEE 802.11 DCF protocol [8]. Therefore, we believe that at channel utilization levels greater than 84%, the transmission of data frames at the lower data rates of 1 or 2 Mbps significantly reduces the throughput of the network. Section 6 presents results that confirm this hypothesis.

From our observations, we believe that the wireless communication channel is *highly congested* when the throughput and goodput of the wireless channel decreases after reaching their respective maximums. For the wireless network at the IETF, we define the network to be highly congested when the channel utilization level is greater than 84%. In this paper we also use the saturation throughput and goodput observations to classify congestion in the communication channel. The variations in link layer behavior, such as the effectiveness of the RTS–CTS mechanism, the number of successfully acknowledged data frames, retransmissions, and the acceptance delay of data frames can be better explained by using congestion classes, as described in the next section.

5.3 Classifying Congestion

In this paper, we suggest that congestion in an IEEE 802.11b wireless network can be classified by using the observed trends in throughput and goodput with respect to increasing channel utilization levels. We classify network states into three classes: uncongested, moderately congested, and highly congested. In the case of the IETF wireless network, an *uncongested* channel is a channel that experiences less than 30% utilization. Since the throughput and goodput of the channel shows a gradual increase from 30% utilization to 84%, the channel is *moderately congested* for utilization values in the range of the 30%–84%. A channel is stated to be *highly congested* when the channel utilization is greater than the 84% threshold.

6 Effects of Congestion

This section discusses the effect of the different congestion levels on network characteristics, behavior of the RTS–CTS mechanism, channel busy-time, reception of frames of different sizes transmitted at different rates, and acceptance delays for data packets. These characteristics offer a basis for understanding the operation of the IEEE 802.11b MAC protocol in heavily congested networks.

To better understand the effects of congestion, we categorize a frame into one of 16 different categories. The categories are defined as a combination of (1) the four possible data rates: 1, 2, 5.5, and 11 Mbps, and (2) the four different frame size classes: small, medium, large and extra-large. The frames are split into the four size classes so that the effect of congestion on different sized frames can be derived separately. The four size classes are defined as follows:

Small (S): frame sizes between 1–400 bytes
Medium (M): frame sizes between 401–800 bytes
Large (L): frame sizes between 801–1200 bytes
Extra-large (XL): frame sizes greater than 1200 bytes

The behavior of the small size class is representative of short control frames and data frames generated by voice and audio applications. The medium, large, and extra-large size class represents the frames generated by file transfer applications, SSH, HTTP, and multimedia video applications.

6.1 RTS–CTS Mechanism

The RTS–CTS mechanism helps reduce frame collisions due to hidden terminals. However, the use of the mechanism is optional. In our data sets we observe that only a small fraction of data frames utilized the RTS–CTS frames to access the channel for transmission. Figure 7 shows that as channel utilization increased, the number of RTS frames increased. Specifically, in the moderate congestion range between 80% and 84% utilization, the average number of RTS frames transmitted per second shows an increase from 5 to 8. This is because, as utilization increases, a greater number of collisions results in a greater number of RTS frames required to access the medium. At the same time, the number of CTS frames does not increase at the same rate because of the failure to receive the RTS frames.

At high congestion levels, the number of RTS frames decreases rapidly because congestion in the medium limits channel access opportunities for their transmission. The number of CTS frames also decreases at high congestion

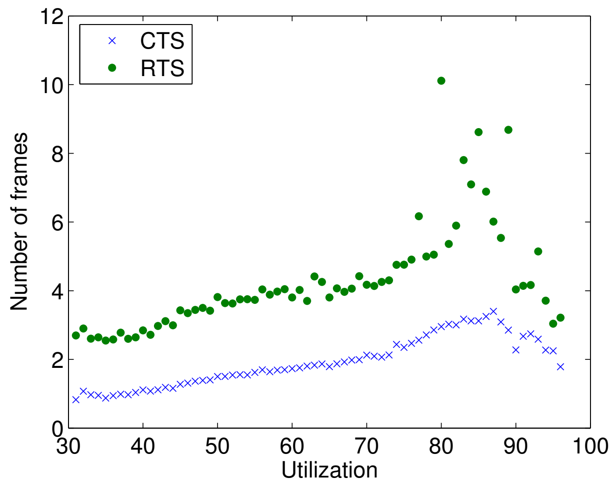


Figure 7: Average number of RTS and CTS frames transmitted per second on the wireless channel versus the channel utilization.

levels because receivers experience a similar limitation for channel access.

When a limited number of devices use the RTS–CTS mechanism, fair channel access for the devices that use the mechanism is also limited. That is because, the devices that utilize the mechanism to transmit DATA frames rely on the successful delivery of the RTS and CTS frames preceding the DATA frame. On the other hand, devices that do not utilize the mechanism solely rely on the successful delivery of the DATA frame. During congestion, this problem is more pronounced because the probability of the delivery of frames decreases due to collisions. Thus, our observations suggest that the use of the RTS–CTS mechanism is deemed to be unfair in congested networks in which only a small set of users depend on the mechanism.

6.2 Channel Busy-Time

Channel busy-time is defined as the fraction of the one second interval during which the channel is either occupied by the transmission of frame bytes or IEEE 802.11b standard specified delays between frame transmissions. In this section, we evaluate the effect of different levels of congestion on the channel busy-time measure for the four different data rates. In Section 5 we observed that, during high congestion, the network throughput and goodput decrease as channel utilization increases. The drop in throughput and goodput can be attributed to the large number of low data rate frames transmitted on the channel. This observation can be better understood by using the trends illustrated in Figure 8.

Figure 8 shows the fraction of a one second interval occupied by 1, 2, 5.5, and 11 Mbps frames at each channel

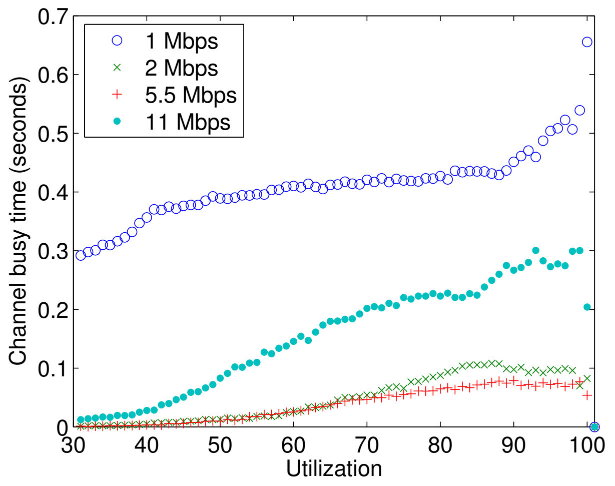


Figure 8: Channel busy-time share of each of the four data rates versus the channel utilization.

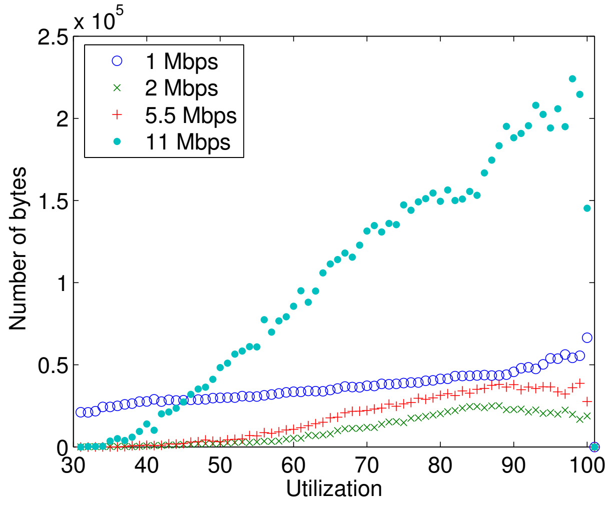


Figure 9: Average number of bytes transmitted per second at each of the four data rates versus the channel utilization.

utilization level. Figure 9 shows the total number of bytes transmitted on the channel per second at each channel utilization level. The figures suggests that the average fraction of a one second period occupied by the 1 Mbps frames is much greater than the time occupied by the frames transmitted at 11 Mbps, even though the number of bytes transmitted at 11 Mbps is significantly greater than the number of bytes transmitted at 1 Mbps at almost all levels of channel utilization. Moreover, during high congestion, the average fraction of one second occupied by 1 Mbps frames increases from 0.43 seconds to 0.54 seconds. As the fraction occupied by the transmission of 1 Mbps frames increases, the throughput and goodput of the network decrease. This confirms our hypothesis that the drop in the throughput and goodput during high congestion is because of the larger

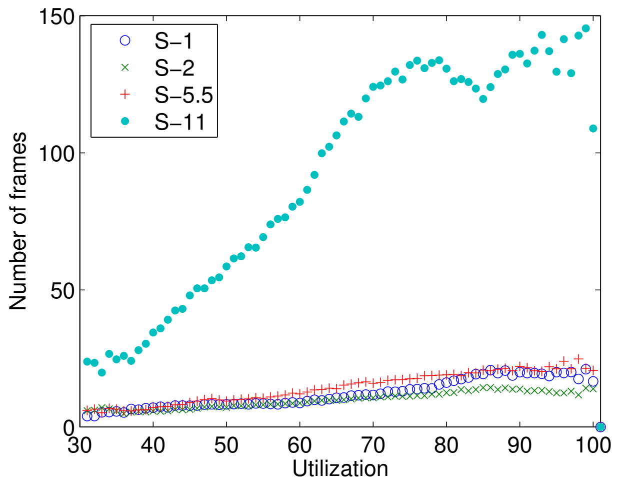


Figure 10: Average number of small data frames transmitted per second on the wireless channel versus the channel utilization.

fraction of time occupied by slower 1 Mbps frames in the network.

6.3 Frame Transmissions

In this section we provide statistics on the number of data frames transmitted on the channel at the four data rates (1, 2, 5.5, and 11 Mbps) and for each frame size class (S, M, L, and XL). The naming convention for the type of frames follows a *size-rate* format. For instance, an S frame transmitted at 11 Mbps is named *S-11* and an XL frame transmitted at 1 Mbps is named *XL-1*.

Figure 10 shows the average number of frames of size S transmitted per second on the channel at each channel utilization. Each point on the graph is an average over our entire data set, including both the day and the plenary session. The number of frames transmitted per second includes both the frames sent at the first attempt and retransmitted frames. We observe that as utilization increases, the number of transmitted S-1, S-2, S-5.5, and S-11 frames increases. However, the number of S-11 frames is significantly larger than the number of frames sent at the other data rates. Cantieni et al. present analytical results that suggest that when an IEEE 802.11b wireless network experiences a state of congestion or throughput saturation, the smaller sized frames sent at the highest rate of 11 Mbps have a higher probability of successful transmission [4] that frames sent at lower rates. In line with these results, we observe a rise in the number of S-11 frames transmitted during high congestion.

Figure 11 shows the average number of XL frames transmitted per second on the channel at each utilization level. We observe that the number of XL-11 frames is greater than the number of frames sent at lower rates. During congestion, the number of XL-11 frames transmitted

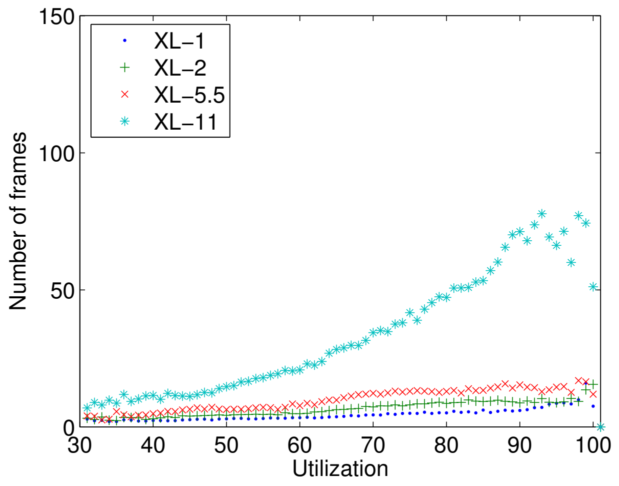


Figure 11: Average number of extra-large data frames transmitted per second on the wireless channel versus the channel utilization.

per second also increases. This increase can be attributed to the increase in the channel access capability of 11 Mbps frames.

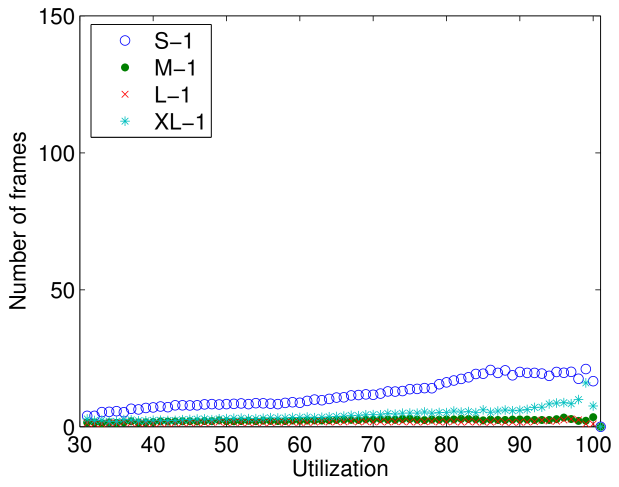


Figure 12: Average number of data frames transmitted per second at 1 Mbps data rate on the wireless channel versus the channel utilization.

Figure 12 shows the average number of frames transmitted at 1 Mbps per second at each channel utilization level. The figure shows that there were a greater number of S-1 frames in the data set compared to the number of XL-1 frames transmitted per second. During high congestion we observe that the number of S-1 and XL-1 frames showed an increase. The increase can be attributed to the multi-rate adaptation algorithms that decrease the sending rate for frame retransmissions.

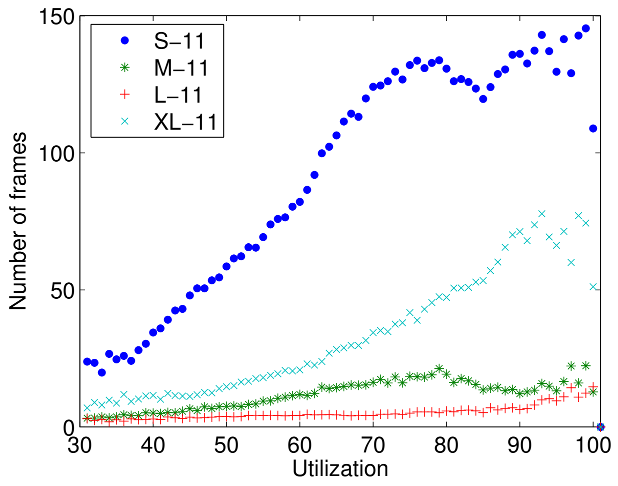


Figure 13: Average number of data frames transmitted per second at 11 Mbps data rate on the wireless channel versus the channel utilization.

Figure 13 shows the average number of frames transmitted at 11 Mbps per second at each channel utilization level. The figure indicates that a large number of data frames are transmitted at the highest data rate. However, during high congestion the number of S-11 and XL-11 frames transmitted per second increases as channel utilization increases. This increase can be attributed to the increase in the number of retransmissions during high congestion.

6.4 Frame Reception

In this section we evaluate the number of successfully acknowledged data frames that were acknowledged at their first attempt of transmission. The evaluation of frame reception includes statistics for S-1, XL-1, S-11 and XL-11 frames. We believe that the evaluation of the behavior of this set of frames is representative of the whole set of results.

A successfully acknowledged data frame is defined as a data frame for which the source receives an acknowledgment frame from the receiving station within an SIFS time delay. In our data set, we identify acknowledged frames as data frames that are immediately followed by an acknowledgment from the receiving station. Other cases include: (1) when the receiving station does not send an acknowledgment because it failed to receive the data frame successfully, (2) the receiving station sends an acknowledgment but the sniffer failed to capture the frame due to either bit errors or the hidden terminal problem, or (3) when the acknowledgment frame from the receiving station was not encountered immediately following the data frame sent by the sending station; the frame is considered to be *not acknowledged* or *dropped*.

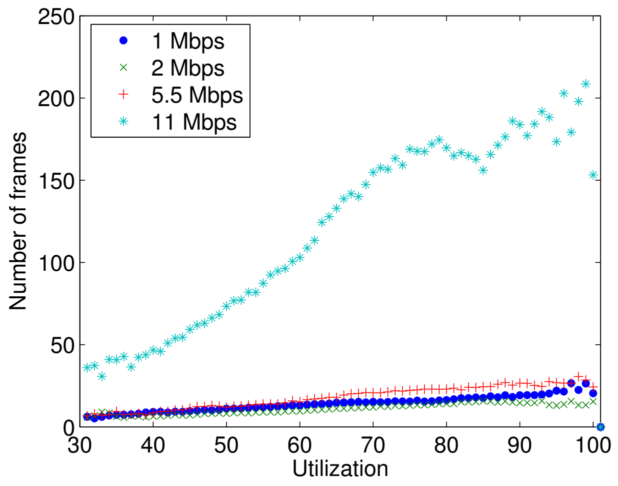


Figure 14: Average number of data frames successfully acknowledged per second at their first attempt of transmission versus the channel utilization.

Figure 14 shows the average number of data frames per second that were acknowledged at their first attempt at transmission for different channel utilization levels. The figure shows that during moderate congestion, there is an increase in the number of 11 Mbps frames acknowledged per second. But, at utilization levels specifically between 80% and 84%, the number of 11 Mbps frames acknowledged per second decreases due to contention in the network. However, during high congestion, the number of 11 Mbps frames that are successfully acknowledged increases. The increase can be attributed to the higher probability of the faster 11 Mbps frames being received as the number of slow 1 Mbps frames transmitted in the network increases.

Thus, our conclusion from this observation is that the reduced sending rate causes a decrease in the throughput achieved during congestion due to larger CBTs of 1 Mbps frames. Also, 11 Mbps frames have a higher probability of reception during high congestion.

6.5 Acceptance Delay

The *Acceptance Delay* for a data frame is the time taken for a data frame to be acknowledged, independent of the number of attempts to transmit. In other words, it is the time computed between the transmission of a data frame and the time when the acknowledgment was recorded. Evaluation of the acceptance delay is significant because it gives us an opportunity to observe the average time taken for a data frame to be delivered and acknowledged at increasing channel utilization levels. Our hypothesis is that the acceptance delay information at different channel utilization levels can be used to make intelligent decisions about choosing data send-rates.

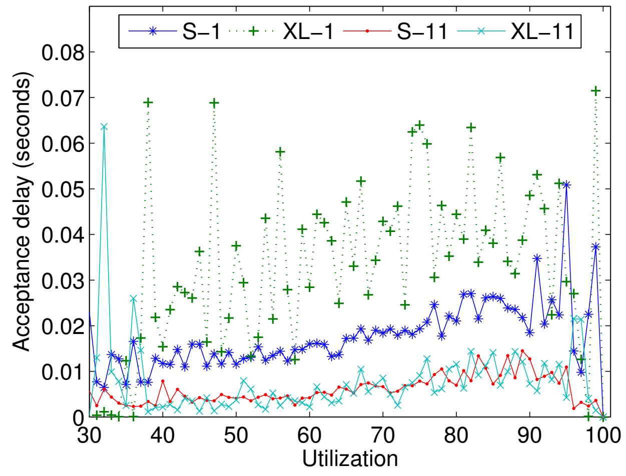


Figure 15: Acceptance delay (in seconds) for data frames successfully acknowledged per second versus the channel utilization.

Figure 15 shows the acceptance delays computed for S-1, S-11, XL-1, and XL-11 frames that were successfully acknowledged during the day and the plenary sessions. We observe a noticeable rise in the acceptance delays as utilization levels increase. However, the acceptance delay values for S-1 and XL-1 frames are significantly greater than the acceptance delays for S-11 and XL-11 frames. The figure also shows that the acceptance delays for S-1 frames are greater than the acceptance delays for XL-11 frames. This observation indicates that the performance of frames that are transmitted at 11 Mbps is better than the performance of frames sent at 1 Mbps, independent of the size of the frames. Additionally, the figure shows that acceptance delays for XL-1 frames are greater than the acceptance delays for S-1 frames. On the other hand, S-11 and XL-11 frames exhibit very similar acceptance delay curves. Therefore, we conclude that the size of a frames transmitted at 1 Mbps has a significant effect on the acceptance delays.

In summary, we hypothesize that for better upper layer protocol performance and to maintain overall network throughput, transmitting data frames at higher rates is better than transmitting frames at lower rates.

7 Conclusions

The analysis of heavily congested wireless networks is crucial for the robust operation of such networks. To this end, this paper has presented an analysis of a large-scale IEEE 802.11b wireless network deployed at the Internet Engineering Task Force meeting in Minneapolis, Minnesota. Specifically, we have investigated the effect of congestion on network throughput and goodput, channel busy-time, the RTS–CTS mechanism, frame transmission and reception, and acceptance delay. However, we believe that the data sets that we collected through vicinity sniffing techniques can be further analyzed to further broaden our scope of understanding.

Observations made in this paper suggest that the use of lower data rates to transmit frames in the network significantly decreases the network throughput and goodput. Therefore, the use of low data rates between two nodes should only be used to alleviate frame losses occurring due to bit errors, low signal-to-noise (SNR) ratio of received frames, or the transmission of frames to greater distances. During congestion, higher data rates should be used. However, the multirate adaptation scheme implemented in commodity radios does not distinguish between frame losses that occur due to any of these causes. Consequently, the response of multirate adaptation schemes to frames losses often results in the poor choice of transmission rates in heavily congested environments. As a result, overall network performance is adversely impacted. Alternate multirate adaptation schemes [9, 18] that determine an optimal packet transmission rate based on SNR may offer some relief. As another strategy to utilize high data rates, clients may choose to dynamically increase or decrease the transmit power of the radio such that data frames can be consistently and reliably transmitted at high data rates.

Another observation made in the paper is the failure of the RTS–CTS mechanism to provide fair channel access to the few nodes using the mechanism. Therefore, during high congestion, the use of the mechanism should be avoided.

8 Acknowledgments

Funding for this work is in part through NSF Network Research Testbeds (NRT) grant ANI-0335302 as part of the WHYNET project and a grant from Nokia Research through the UC Micro program. We would like to thank Jim Martin (Netzwert AG), Karen O Donoghue (Naval Surface Warfare Center), Bill Jensen (University of Wisconsin), Jesse Lindeman (Airespace), and the IETF62 NOC Team for assisting us in the collection of the data used in this paper.

References

[1] I. Aad, Q. Ni, C. Barakat, and T. Turletti. Enhancing IEEE 802.11 MAC in Congested Environments. In *Proceedings of IEEE ASWN*, Boston, MA, August 2004.

[2] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing User Behavior and Network Performance in a Public Wireless LAN. In *Proceedings of ACM SIGMETRICS*, pages 195–205, Marina Del Rey, CA, June 2002.

[3] M. Balazinska and P. Castro. Characterizing Mobility and Network Usage in a Corporate Wireless Local-area Network. In *Proceedings of ACM MOBICOM*, San Francisco, CA, May 2003.

[4] G. R. Cantieni, Q. Ni, C. Barakat, and T. Turletti. Performance Analysis under Finite Load and Improvements for Multirate 802.11. *Elsevier Computer Communications Journal*, 28(10):1095–1109, June 2005.

[5] S. Cen, P. C. Cosman, and G. M. Voelker. End-to-end Differentiation of Congestion and Wireless Losses. *ACM/IEEE Transactions on Networking*, 11(5):703–717, October 2003.

[6] F. Chinchilla, M. Lindsey, and M. Papadopouli. Analysis of Wireless Information Locality and Association Patterns in a Campus. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[7] T. Henderson, D. Kotz, and I. Abyzov. The Changing Use of a Mature Campus-wide Wireless Network. In *Proceedings of ACM MOBICOM*, Philadelphia, PA, September 2004.

[8] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance Anomaly of 802.11b. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, March–April 2003.

[9] G. Holland, N. Vaidya, and P. Bahl. A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks. In *Proceedings of ACM MOBICOM*, pages 236–251, Rome, Italy, July 2001.

[10] A. Jardosh, K. Ramachandran, K. Almeroth, and E. Belding-Royer. Understanding Link-Layer Behavior in Highly Congested IEEE 802.11b Wireless Networks. In *Proceedings of ACM SIGCOMM Workshop E-WIND*, Philadelphia, PA, August 2005.

[11] J. Jun, P. Peddabachagari, and M. Sichitiu. Theoretical Maximum Throughput of IEEE 802.11 and its Applications. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, pages 249–257, Cambridge, MA, April 2003.

[12] A. Kamerman and L. Monteban. WaveLAN 2: A High-performance Wireless LAN for the Unlicensed Band. In *Bell Labs Technical Journal*, Summer 1997.

[13] D. Kotz and K. Essien. Analysis of a Campus-wide Wireless Network. In *Proceedings of ACM MOBICOM*, Atlanta, GA, September 2002.

[14] X. Meng, S. Wong, Y. Yuan, and S. Lu. Characterizing Flows in Large Wireless Data Networks. In *Proceedings of ACM MOBICOM*, Philadelphia, PA, September 2004.

[15] A. Mishra, M. Shin, and W. A. Arbaugh. An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process. *ACM SIGCOMM Computer Communication Review*, 33(2):93–102, 2003.

[16] E. Modiano. An Adaptive Algorithm for Optimizing the Packet Size Used in Wireless ARQ Protocols. *Wireless Networks*, 5(4):279–286, July 1999.

[17] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan. Measurement-based Characterization of 802.11 in a Hotspot Setting. In *Proceedings of EWIND*, Philadelphia, PA, August 2005.

[18] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic Media Access for Multirate Ad Hoc Networks. In *Proceedings of ACM MOBICOM*, pages 24–35, Atlanta, USA, September 2002.

[19] D. Schwab and R. Bunt. Characterizing the Use of a Campus Wireless Network. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[20] Y. Tay and K. Chua. A Capacity Analysis for the IEEE 802.11 MAC Protocol. *Wireless Networks*, 7(2):159–171, March 2001.

[21] M. Torrent-Morenoe, D. Jiang, and H. Hartenstein. Broadcast Reception Rates and Effects of Priority Access in 802.11 Based Vehicular Ad hoc Networks. In *Proceedings of ACM VANET*, Philadelphia, PA, October 2001.

[22] C. Tuduce and T. Gross. A Mobility Model Based on WLAN Traces and its Validation. In *Proceedings of IEEE INFOCOM*, Miami, FL, March 2005.

[23] J. Yeo, M. Youssef, and A. Agrawala. A Framework for Wireless LAN Monitoring and its Applications. In *Proceedings of the ACM Workshop on Wireless Security*, pages 70–79, Philadelphia, PA, October 2004.

Notes

[1] http://www.ietf.org/

[2] http://www.cisco.com/en/US/products/ps6306/index.html

Facilitating Access Point Selection in IEEE 802.11 Wireless Networks

S. Vasudevan[†], *K. Papagiannaki*[‡], *C. Diot*[‡], *J. Kurose*[†] *and D. Towsley*[†]

[†]*Dept. of Computer Science,*
University of Massachusetts,
Amherst, MA 01003
{svasu,kurose,towsley}@cs.umass.edu

[‡]*Intel Research*
Cambridge, UK
CB3 0FD
{dina.papagiannaki,christophe.diot}@intel.com

Abstract

The performance experienced by wireless clients in IEEE 802.11 wireless networks heavily depends on the clients' ability to identify the Access Point (AP) that will offer the best service. The current AP affiliation mechanism implemented in most wireless clients is based on signal strength measurements received by the client from all the APs in its neighborhood. The client then affiliates with the AP from which it receives the strongest signal. It is well-known that such an algorithm can lead to sub-optimal performance, due to its ignorance of the load at different APs.

In this work, we consider the problem of AP selection. We identify *potential bandwidth* as the metric based on which hosts should make affiliation decisions, and define it as the (MAC-layer) bandwidth that the client is likely to receive after affiliating with a particular AP. We further limit ourselves to the use of passive measurements that do not require an end-host to affiliate with the AP, thus allowing the end-host to simultaneously evaluate the potential bandwidth to multiple APs in range. This can also facilitate more informed roaming decisions. We propose a methodology for the estimation of potential upstream and downstream bandwidth between a client and an AP based on measurements of delay incurred by 802.11 Beacon frames from the AP. Preliminary experiments conducted in a controlled environment demonstrate that the proposed methodology looks promising, yielding fairly accurate results under varying conditions.

1 Introduction

There has been an enormous growth in the adoption of IEEE 802.11 wireless networks in the last few years. The ease of installation and the low infrastructure cost of 802.11 networks makes them ideal for network access in offices, malls, airports, cafes, hotels and so on. The widespread deployment of IEEE 802.11 networks means that a wireless client is often in the vicinity of multiple APs with which to affiliate. The selection of the AP that the client decides to affiliate with needs to be done carefully since it will dictate the client's eventual performance.

The conventional approach to access point selection is based on received signal strength measurements from the access points within range. However, it has been pointed out in several papers [1, 2, 4] that affiliation based on signal strength can lead to very bad performance for the end-host, since the signal-strength metric does not convey information regarding other attributes that affect end-host performance, such as the AP load and the amount of contention on the wireless medium.

In this paper, we describe how an end-host can take the aforementioned attributes into account while choosing an access point to affiliate with. We identify a metric that can capture *the bandwidth that an end-host is likely to receive if it were to affiliate with a given access point*, which we call *potential bandwidth*. The MAC-layer bandwidth offered by different wireless networks in the vicinity of the wireless client is a desirable metric as it takes into account the AP load, the contention on the wireless medium, as well as the signal strength.

In designing an affiliation algorithm based on potential bandwidth estimation, several constraints must be taken into consideration. The algorithm needs to be non-intrusive, i.e. it should not introduce additional overhead to the APs or their affiliated users. The algorithm should not require any changes at the AP side, if possible. More importantly, such an algorithm should be able to estimate potential bandwidth without previous affiliation with the several APs within range. Such a constraint minimizes the amount of time a client spends in the evaluation of the several choices it may have (since it does not associate and dis-associate with the different APs), while allowing for the continuous evaluation of AP performance even when an affiliation has taken place[1]. The latter implies that a wire-

[1]Notice that in the case of the initial affiliation the client will be able to identify the AP that offers the highest potential bandwidth on *any* frequency. In the case of roaming, the client will be able to quantify potential

less client implementing the proposed functionality will be able to make more informed and efficient roaming decisions, continuously quantifying the performance of all APs in range.

In this paper, we propose a methodology for the estimation of potential bandwidth between a given AP and an end-host that fulfills the aforementioned requirements. The proposed methodology does not require the end-host to change its current affiliation and introduces very little overhead. Unlike [1, 2], the affiliation algorithm proposed in this paper is end-host initiated and therefore, does not necessitate changes at the AP.

In a nutshell, our approach to potential bandwidth estimation relies on passive measurements of the timings of beacon frames sent out by an AP. Beacon frames are broadcast by APs periodically, and are used by APs to announce their identity as well as for the synchronization of the entire network. The delay between the time when a beacon frame is scheduled for transmission and its eventual transmission captures the load of the AP and the contention inside the network, conditions that the client would face if affiliated with that AP. The corresponding delay of data frames provides an estimate for the bandwidth a client will receive from the AP downstream. Upstream potential bandwidth estimation relies on frames sent by the client to the AP in the unaffiliated state and is based on a similar methodology that quantifies the respective delays.

Our technique can be used as part of an AP selection mechanism or for the evaluation of a wireless network's health. We evaluate its accuracy using controlled experiments in a low-noise environment. Preliminary experiments indicate that our approach yields fairly accurate estimates of the actual bandwidth from the AP to end-host, indicating that our approach looks promising.

The rest of the paper is structured as follows. In the next section, we describe related work. In Section 3, we describe our potential bandwidth estimation scheme. We discuss experimental results in Section 4. Finally, we conclude and describe in detail future directions in Section 5.

2 Related Work

The conventional AP selection mechanism, based on signal strength measurements, has been shown to lead to poor user experience [1, 2, 7] and highly unbalanced load distribution among APs [4]. Due to these shortcomings there have been several alternative proposals which typically fall in one of three categories: (i) AP-assisted [3, 7, 1], (ii) centralized [2], and (iii) active [8] solutions. In this work we take a step back and look at the fundamental requirements of the AP selection problem. Based on the identified requirements, we propose a technique that *does not require the assistance of the AP*, *does not require previous affiliation of the client with an AP*, and *is initiated by the client without the need for central coordination*. Such properties allow for the continuous evaluation of the "quality" of all APs within range that could also facilitate better roaming decisions.

bandwidth only for the APs residing in the *same* frequency.

Our work targets the estimation of the potential bandwidth and not the available bandwidth as in [5], which is defined as the maximum rate at which a host can send its data without lowering the sending rates of other already affiliated hosts. In this work, we are not interested in the bandwidth available to a client before affiliation, but the MAC-layer bandwidth the client will receive after it affiliates with the AP. In addition, we do not aim to estimate the layer-3 throughput that a client would receive once affiliated with an AP, since such an estimation would require knowledge of the client's workload and its path through the wired network. The metric of potential bandwidth can characterize the wireless part of the client's connections. In future work, we intend to look into passive measurement techniques that could allow us to extend our estimates to account for the wired part of the network, say by passively observing the performance currently experienced by other users in the same wireless network.

The closest recent work to ours is [6], where the authors propose a methodology for passive bandwidth estimation between two communicating wireless stations. However, their method does not provide an estimate of the potential bandwidth that an end-host is likely to receive on a wireless link with another host (when one of the hosts is not part of the network yet).

3 Potential Bandwidth Estimation

In this section, we describe how an end-host can estimate both the potential upstream and downstream bandwidth between the AP and itself. The final affiliation decision made by the end-host is going to be some function of the upstream and downstream bandwidth and is likely to depend on the user's requirements. For the remainder of this work, we assume that the client has credentials to associate with any AP within range and selects the AP offering the highest bandwidth in the direction the client will use for its data transfer. We begin by providing a brief background of the IEEE 802.11 MAC protocol for data transmission.

3.1 Background

The protocol for data transmission is the same regardless of whether the transmitter is an AP or an affiliated host. Each node (including the AP) that has data to transmit in an IEEE 802.11 network first senses the channel for a duration equal to $DIFS$ (Distributed Inter-Frame Sequence). If the node determines the channel to be idle for this duration, then the node enters a back-off stage, in which it delays its transmission by a random number of time slots (each slot of duration $SLOT$) chosen from the interval $[0, CW]$, where CW is called the contention window size. If the channel is still idle at the end of the back-off stage, then the node transmits a Request-to-Send (RTS) frame to the intended receiver. On receiving the RTS frame, the receiver responds back with a Clear-to-Send (CTS) frame to the sender after a delay equal to Short Inter-Frame Sequence ($SIFS$). Nodes, other than the sender or the receiver, that hear either the RTS or the CTS frame delay their transmissions until after the end of the data transmission between the sender and the receiver, as specified in the duration field of the RTS and CTS frames. Upon receiving the CTS frame, the sender waits for a duration of $SIFS$ and sends its data frame. Finally, the receiver responds back with an ACK frame to acknowledge the receipt of $DATA$ frame. The absence of either a CTS or ACK frame causes the sender to timeout and re-transmit the RTS frame or the $DATA$ frame respectively. Many implementations also allow nodes to simply turn on or disable the RTS/CTS handshake. In this case, nodes directly transmit their data frames, on determining the channel to be idle at the end of the backoff stage.

We first describe our methodology to estimate the downstream bandwidth from an AP to an end-host in the absence of RTS/CTS handshake and then describe how the RTS/CTS handshake mechanism can be accommodated into the estimation scheme. We also discuss how an end-host can determine its upstream bandwidth to an AP. We initially ignore losses and subsequently, describe how losses can be accounted for in Section 3.5.

3.2 Beacon Delays

In order to estimate the downstream bandwidth from the access point to an end-host, we propose a methodology that allows the end-host to estimate the delays of the periodic *Beacon* frames sent from an access point. Figure 1 illustrates how beacon frame transmissions are handled at an access point. As seen from the figure, an access point schedules a *Beacon* frame every *beacon interval* (typically, 102.4 ms). The time instant at which the access point schedules the next beacon message is referred to as the *Target Beacon Transmission Time* ($TBTT$). As per the 802.11 standard, time zero is defined to be a $TBTT$. Given the value of the beacon interval, the end-host knows the exact time instants when beacon messages are scheduled for transmission. Once a beacon message is scheduled, it is transmitted according to the normal frame transmission rules. In this paper, we assume that beacon frames are not prioritized over other frames, as implemented in the APs used in our experiments. Handling beacon frame prioritization is an interesting extension and will be considered in our future work. The time difference between the instant when a beacon message transmission begins (as obtained from the timestamp field of the *Beacon* frame) and the $TBTT$ yields an estimate of the beacon delay, T_B, which is the total time spent by a beacon frame at the access point waiting for transmission. Since we assume that beacon frames are not prioritized over other frames, T_B provides an estimate of the total queuing delay plus the contention delay that will be experienced by a data frame transmitted by the AP. Note that beacon delays are computed solely based on timestamps provided by the access point and thus, synchronization issues do not arise.

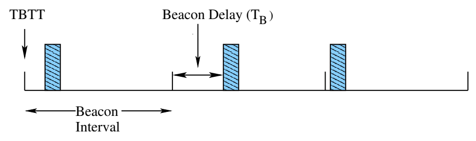


Figure 1: Beacon Transmissions at an Access Point

We now proceed to describe how we can use observed beacon delays to estimate the downstream bandwidth from an AP to a mobile host.

T	total delay incurred by a data frame from an AP
T_D	delay incurred between the instant when a data frame is scheduled for transmission to the instant that the frame is received at the receiver
T_A	delay of the ACK frame from the receiver to the sender
T_B	total contention delay experienced by a data frame from the AP
$DATA$, RTS, CTS	size of the data, RTS, CTS frame respectively
R	data rate at which the sender transmits the data frame
Rb	basic rate at which control frames are transmitted
B	potential bandwidth from the AP to the end-host

Table 1: Notations for the computation of downstream bandwidth

3.3 Downstream Bandwidth estimation in the absence of RTS/CTS

The total delay incurred by a data frame from an AP in the absence of RTS/CTS handshake is given by: the contention and transmission delay of the data frame plus the respective ACK delay.

$$T = T_D + T_A \quad (1)$$

T_D in turn can be estimated from the beacon delay T_B, estimated as in Section 3.2, and the transmission delay of the frame[2], and is given by:

$$T_D = T_B + \frac{DATA}{R} \qquad (2)$$

Upon receiving the data frame, the receiver sends an ACK frame after a delay of $SIFS$. ACK frames are fixed in length and are typically sent at the same rate as the data frame. Hence, knowing the sender rate, T_A can be easily determined as:

$$T_A = SIFS + \frac{ACK}{R} \qquad (3)$$

The potential bandwidth B from the AP to the end-host is then given by:

$$B = \frac{DATA}{T} \qquad (4)$$

3.4 Downstream Bandwidth estimation in presence of RTS/CTS

With the RTS/CTS handshake, each data frame transmission incurs a total delay (T) given by Eq. (5), the sum of delays incurred by the RTS, CTS, data and ACK frames respectively.

$$T = T_R + T_C + T_D + T_A \qquad (5)$$

Since the frame transmission rules for an RTS and beacon frames are the same, the delay incurred by an RTS frame can be estimated using Eq. (6), as the sum of T_B and transmission delay (all MAC control frames are transmitted at the base rate).

$$T_R = T_B + \frac{RTS}{R_b} \qquad (6)$$

Upon receiving a RTS frame, a receiver waits a duration of time equal to $SIFS$ and transmits a CTS frame, again at the base rate R_b. The CTS frame is transmitted at the base rate R_b and its delay is given by:

$$T_C = SIFS + \frac{CTS}{R_b} \qquad (7)$$

The delay incurred by the data frame is given by:

$$T_D = SIFS + \frac{DATA}{R} \qquad (8)$$

Lastly, the computation of T_A remains the same across both schemes and is given by Eq. (3). The potential bandwidth B is then obtained using Eq. (4).

[2]If the AP has multi-rate support, then the current sending rate R of the AP can easily be inferred from the duration fields in the data frames transmitted by the AP.

3.5 Loss Probability Estimation

So far, the potential bandwidth estimation methodology assumed no packet losses. Losses occur due to collisions when multiple wireless stations transmit simultaneously and also due to environmental effects such as multipath, fading etc. Packet losses reduce the bandwidth between communicating stations, since they cause nodes to double their contention window and thereby, backoff for longer durations before retransmitting their data.

Thus, in order to estimate the potential downstream bandwidth from a given AP, an end-host needs to estimate the loss rate on the wireless link from the AP to itself. We propose that nodes infer frame losses, by exploiting the 12-bit sequence number field present in the 802.11 data and management frames. An end-host passively monitors all frames transmitted by the AP for a certain duration. The end-host can then infer data frame losses based on gaps in sequence numbers during the monitoring period. It is possible that the monitoring node may hear a data frame from an AP that is a retransmission of an earlier frame, which it did not hear. In this case, the monitoring node can detect retransmissions by looking at the Retry bit in the Frame Control field of the received frame. If this bit is set, it indicates that the frame is a retransmission of an earlier frame. Since the Retry bit does not indicate the number of retransmissions of a frame, we make a simplifying assumption that the probability of more than two successive retransmissions of a frame between an AP and a host affiliated to that AP is negligible.

The above described method of inferring loss rate, is useful both in the presence of RTS/CTS and in its absence. In the presence of RTS/CTS, the probability of an RTS frame loss differs from the probability of a data frame loss, since an RTS frame is transmitted at the base rate. An RTS frame loss can be inferred by a monitoring end-host, if the monitoring host overhears a data frame transmission from an AP to an end-host, but does not hear the RTS frame transmission from the AP to the end-host preceding the data transmission. Just as in the case of a data frame, an RTS frame retransmission can be detected from the Retry bit in the Frame Control field of the frame. Data frame losses can be detected from the missing sequence numbers over the monitoring period.

The estimated loss probability can be used to calculate the expected delays incurred by the RTS frames and data frames transmitted by an AP. For simplicity, we assume that CTS and ACK frames from the end-host to the AP are transmitted loss-free. This may be a reasonable assumption since CTS and ACK frames are very short. Furthermore, CTS frames are transmitted at the base rate and the ACK frames are transmitted collision-free. This assumption means that CTS and ACK frames always incur fixed delays. Losses then only impact the RTS and data frames in our model. The estimated loss probability can easily be incorporated to obtain the expected back-off delay and the corresponding frame delay, using the analysis shown in [6].

When there are no affiliated hosts, a monitoring node does not overhear any transmissions except the beacon frames transmitted by an AP. Absence of a beacon frame in a *beacon interval* indicates that the beacon frame was lost. A monitoring host can estimate the loss probability of data frames to be the loss probability of the beacon frames. The RTS frames are transmitted at the base rate and can be assumed to be transmitted loss-free, especially given that there is no contention for the medium and that the probability of a collision is zero.

3.6 Upstream Bandwidth Estimation

Our proposed approach to estimating the upstream bandwidth requires that the end-host sends data frames to an access point in the unaffiliated state and records the time elapsed between the instant when a frame is scheduled for transmission and the time when the end-host receives an ACK message. It is interesting to note that the IEEE 802.11 standard allows a station in an unassociated state to send data frames to an access point. By sending several such frames and measuring the delays incurred by the frames, an end-host gets an estimate of the expected delay of a data frame. The potential upstream bandwidth can then be estimated using Eq. (4).

The implementation of the upstream bandwidth estimation scheme requires modifications to the wireless driver to allow a station to send frames in the unaffiliated state and is currently being investigated.

4 Experimental Results

In this section, we describe results from controlled experiments of our downstream bandwidth estimation scheme. All our experiments were conducted in an *anechoic chamber* that is designed to provide a very low noise environment, suitable for controlled experimentation. We configured a linux box with a Netgear MA 311 wireless PCI card to function as an access point running the *hostap* driver. The RTS/CTS handshake was disabled and the card was operated at a fixed rate of 11 Mbps.

4.1 Beacon delays in contention-free environments

In a contention-free environment and when the AP has no load, the mean beacon delay can be expressed as: Mean Beacon Delay = $DIFS + E[CW_{min}] \times SLOT + PLCP$, where $DIFS$ is the duration for which an AP senses the channel before transmitting a beacon frame; $E[CW_{min}] \times SLOT$ is the back-off delay once the AP has sensed the channel to be idle for a duration $DIFS$; and $PLCP$ is the Physical Layer Convergence Protocol overhead associated with every transmitted frame. The IEEE 802.11b standard specifies the various parameter values as follows: $DIFS = 50\mu s$, $SLOT = 20\mu s$, $CW_{min} = 31$, $PLCP = 192\mu s$. From these values, we obtain the mean beacon delay to be 552 μs.

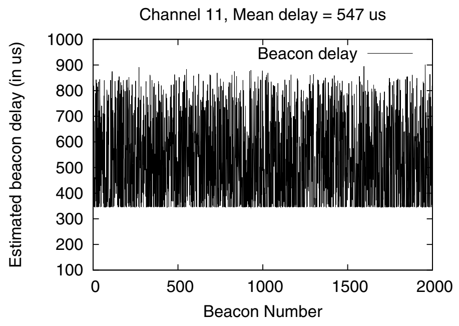


Figure 2: Beacon Delays when the AP has no load

We conduct a number of experiments to estimate the beacon delays using the methodology described in Section 3. Figure 2 shows that the mean estimated beacon delay value is $547\mu s$, which is close to the expected value of $552\mu s$. We next perform experiments to determine whether the bandwidth estimated through the beacon delay measurements closely approximates the actual bandwidth obtained by the end-host upon affiliation with the AP.

4.2 Bandwidth Estimation

In a collision-free environment, we know from Section 4.1 that the mean beacon delay is 552 μs. For a packet of size L bytes and data rate R, the potential downstream bandwidth is then given by (Eq. 4):

$$B = \frac{8L}{552 + \frac{8L}{R} + T_A}$$

where $T_A = 213\mu s$. For instance, when $L = 640$ and $R =$11 Mbps, the potential downstream bandwidth yields an estimate $B =$ 4.16 Mbps.

We performed a simple experiment to verify whether the actual bandwidth observed on the downlink from AP to an end-host compares with the estimated value obtained above. A UDP session is initiated from the AP to an affiliated end-host. The duration of the transfer was 200 seconds and the AP was constantly backlogged. The actual bandwidth B_m from the AP to the end-host for the duration of the transfer was measured to be 4.3 Mbps, which closely agrees with the estimate B obtained above.

In a second experiment, we place one AP and two wireless hosts $H1$ and $H2$ in the anechoic chamber. Host $H1$ is affiliated to the AP. A UDP session is initiated from the AP

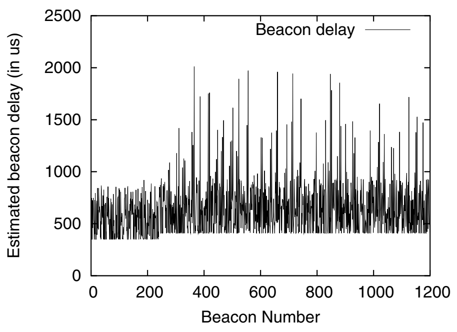


Figure 3: Beacon Delays when the AP is loaded

to the host $H1$. The UDP session consists of CBR traffic generated at the rate of 100 packets/second, each packet of size 576 bytes (640 bytes including the headers). Another host $H2$ is configured in "monitor" mode and records the delays observed for the beacon frames (beacons numbered 300 and higher in Figure 3). The monitoring host $H2$ estimates the mean beacon delay from the AP over the duration of the UDP transfer to be $687\mu s$. Using our bandwidth estimation methodology, $H2$ estimates the potential bandwidth from the AP to itself to be $B =$3.74 Mbps. We then affiliate $H2$ with the AP and initiate another UDP session between the AP and host $H2$, simultaneously with the UDP session between the AP and $H1$. The AP is always backlogged with packets for $H2$. The actual bandwidth B_m from the AP to $H2$ is measured to be 4.06 Mbps, which agrees with the potential bandwidth estimate of 3.74 Mbps, obtained by $H2$ prior to affiliation with the AP. Thus, the experimental results suggest that our approach is promising.

5 Conclusions and Future Work

In this paper, we argued for potential bandwidth between AP and end-host as an important metric in the process of AP selection. We described a methodology for estimating the potential bandwidth based on delays experienced by beacon frames from an AP. We also presented results from experiments conducted in a low-noise environment, which showed that the proposed approach yields fairly accurate estimates of the bandwidth.

The described work is in progress and is currently looking at the following issues: - In this paper, we showed results from experiments in a noise-free environment. We plan to evaluate our bandwidth estimation scheme in the presence of noise, using controlled experiments. - The issue of how frequently should nodes estimate bandwidth to various APs in range and the duration over which estimations need to be carried out is currently under investigation. - We evaluate our bandwidth estimation with Netgear MA 311 Wireless cards that use the DCF (Distributed Co-ordination Function) mode of operation. Extending our scheme to be able to estimate the bandwidth between PCF-based (Point Co-ordination Function) APs and end-hosts is another interesting future question. - In the Netgear 311 wireless cards, the beacon frames were transmitted with the same priority as the data frames. We wish to consider the case, when beacons are prioritized over other frames. - Finally, our estimation depends on the assumption that time zero at the AP is the time instant when the first beacon frame is scheduled for transmission, as specified in the IEEE 802.11 standard. While we observed this is very likely the case with the Netgear cards we experimented with,different vendors can be expected to implement beaconing differently. Inferring TBTTs by observing inter-beacon times remains a topic for further investigation.

References

[1] A. Balachandran, P. Bahl, and G. Voelker. Hot-spot congestion relief and service guarantees in public-area wireless networks. *SIGCOMM Computer Communication Review*, 32(1), 2002.

[2] Y. Bejerano, S. Han, and L. Li. Fairness and load balancing in wireless LANs using association control. In *Proceedings of ACM Mobicom*, Philadelphia, Oct 2004.

[3] Cisco Systems Inc. Data sheet for cisco aironet 1200 series, 2004.

[4] G. Judd and P. Steenkiste. Fixing 801.11 access point selection. In *Poster in Proceedings of ACM Mobicom*, Pittsburgh, Aug 2002.

[5] K. Lakshminarayanan, V. Padmanabhan, and J. Padhye. Bandwidth estimation in broadband access networks. In *Proceedings of ACM Internet Measurements Conference*, Taormina, Oct 2004.

[6] S. Lee, S. Banerjee, and B. Bhattacharjee. The case for a multi-hop wireless local area network. In *Proceedings of IEEE Infocom*, Hong Kong, Mar 2004.

[7] R. Murty and E. Qi. An adaptive approach to wireless network performance optimization. Technical report, Corporate Technology Group (CTG), Intel Corporation, Technical Report, 2004.

[8] S. Shah, K. Chen, and K. Nahrstedt. Available bandwidth estimation in IEEE 802.11-based wireless networks. In *Proceedings of 1st ISMA/CAIDA Workshop on Bandwidth Estimation (BEst)*, San Diego, Dec 2003.

Eliminating handoff latencies in 802.11 WLANs using Multiple Radios: Applications, Experience, and Evaluation

Vladimir Brik, Arunesh Mishra, Suman Banerjee [*]

Abstract

Deployment of Voice-over IP (VoIP) and other real-time streaming applications has been somewhat limited in wireless LANs today, partially because of the high handoff latencies experienced by mobile users. Our goal in this work is to eliminate handoff latency by exploiting the potential of multiple radios in WLAN devices. Our proposed approach, called *MultiScan,* is implemented entirely on the client-side, and, unlike prior work, MultiScan requires neither changing the Access Points (APs), nor having knowledge of wireless network topology. MultiScan nodes rely on using their (potentially idle) second wireless interface to opportunistically scan and pre-associate with alternate APs and eventually seamlessly handoff ongoing connections. In this paper we describe our implementation of MultiScan, present detailed evaluations of its effect on handoff latency and evaluate performance gains for MultiScan-enhanced wireless clients running Skype, a popular commercial VoIP application.

1 Introduction

IEEE 802.11 [1] based wireless LAN (WLAN) technologies have been experiencing an unprecedented growth in the recent years fueled partly by decreasing costs and increasing data rates available through them. From the users' perspective, the key advantage of such networks is untethered access: users can freely move within their area of coverage and stay connected.

In 802.11 WLANs clients connect to the Internet via Access Points (APs). Due to design choices and requirements of the 802.11 standard, the communication range of 802.11 devices is rather limited, and it is not uncommon for an AP to have an effective communication range of less than 60 meters indoors. Consequently, WLAN coverage over a large geographic area is provided using multiple APs, and a wireless client moving through such area is likely to pass from the coverage area of one AP to that of another. In order to maintain continuous connectivity, the mobile client has to switch between APs in a process known as a *handoff.* For mobile clients handoffs can occur very often because of the short range of APs.

[*]V. Brik and S. Banerjee are with the Department of Computer Sciences, University of Wisconsin-Madison, WI 53706, USA. Email: {vladimir,suman}@cs.wisc.edu. A. Mishra is with the Department of Computer Science, University of Maryland, College Park, MD 20742, USA. Email: arunesh@cs.umd.edu

The 802.11 standard does not completely specify the handoff procedure. Depending on the hardware and the vendor, it may take between 60 ms and 400 ms (about 260 ms on average) to complete a handoff, and in some cases a node may experience a connectivity gap of up to a second [7, 11]. Such high handoff latencies are adequate for *discrete mobility* scenarios where a client (typically a laptop user) uses the network while stationary, then moves to a different location but does not use the network during the move, and resumes network usage when stationary again. However, such handoff performance is highly inadequate for *continuous mobility* scenarios, where a client needs to use the network while mobile through a sequence of handoffs. Continuous mobility scenarios are of great significance in real-time latency-sensitive applications, e.g., Voice-over-IP (VoIP) and other synchronous multimedia applications. Poor handoff performance is one of the major hindrances to deployment of VoIP applications in WLANs.

The goal of this paper is to address the need for seamless mobility in WLANs, thereby meeting the needs of VoIP and other latency-sensitive applications. We propose a solution called MultiScan that uses two 802.11 network interfaces on the same device (e.g. an 802.11-based wireless phone or a PDA). Our experimental results demonstrate that MultiScan is capable of *completely eliminating* handoff latencies, and to our knowledge, it is the only existing practical approach that can eliminate handoff latency.

Why do we need two radios?

We believe that a two-radio interface solution is both practical and feasible, and is the only mechanism that can eliminate handoff latencies in WLANs. While two physically separate radio interfaces in a single device may seem impractical (especially in a small form factor), it turns out that multiple commercial vendors are coming out with multi-band chipsets that allow communication on two or more channels, e.g., EN-3001 intelligent wideband WLAN chipset for 802.11 networks (see http://www.engim.com/). Hence, we believe that approaches to handoffs that are based on using two radio interfaces are both practical and timely, and can jumpstart the process of efficient deployment of VoIP applications in WLANs.

Minimization of handoff latency for single-radio WLAN clients has been examined in prior research, e.g., Neighbor Graphs [3] and SyncScan [2]. In the Neighbor Graphs approach, extra functionality is implemented at both clients

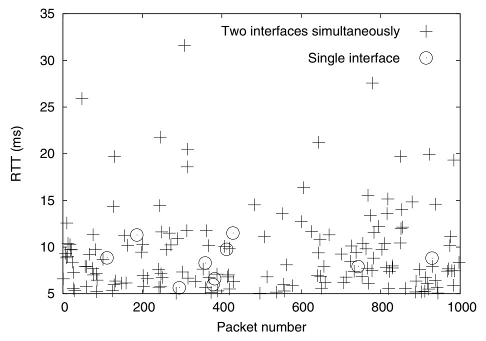


Figure 1: Performance of two radios used simultaneously.

and APs that is used to infer WLAN topology and reduce handoff overhead to around 30-40 ms. In the SyncScan approach, all clients and all APs in the network require time synchronization. While SyncScan potentially reduces the latency of a handoff to a few milliseconds, it requires regular suspension of communication for roughly twice the amount of time it takes a wireless interface to change channel, which, depending on hardware, could exceed 10 ms.

An important advantage of MultiScan is that it requires changes *only* at the wireless client. In particular, we have implemented all the necessary functionality in a (Linux) client as a kernel module that controls the handoff process and (re)association decisions. The MultiScan module relies only on the standard Linux kernel API and does not depend on device drivers. Therefore, MultiScan is completely compatible with any wireless card. Finally, MultiScan's operation takes place only in the link layer and hence is transparent to the applications. MultiScan module will be made publically available shortly.

In this paper we present the design of MultiScan, its prototype implementation, and demonstrate the usefulness of our approach for seamless handoffs in 802.11 WLANs. In particular, we evaluate the performance of MultiScan using traffic floods and a popular commercial VoIP application Skype [15].

Exploiting multiple radios

Wireless nodes with multiple radios have more resources at their disposal than their single-radio counterparts. However, naive use of these additional resources could significantly hurt performance. This was demonstrated in a work by Adya et al. [8] in the context of multi-radio mesh networks, where wireless nodes are equipped with multiple radio interfaces and traffic takes multi-hop wireless paths through them. A possible use of two radio interfaces is to use them *simultaneously,* i.e., on every wireless hop, each node will use its two radios to form two wireless links with its neighboring node (also equipped with two radios) operating on different channels with data traffic striped across the two links. Adya et al. showed that when two such links between a pair of nodes are used simultaneously, TCP-based applications perform poorly due to re-ordering effects, unless the loads of the two links are well balanced.

In our work we have found that using two radio interfaces simultaneously in a single device, especially those with small form factors (such as a PDA or a handset), leads to significant loss of performance due to cross-interference between the radios. This is true even if the two interfaces are operating on different 802.11 wireless channels and occur due to the physical proximity of transceiver circuitry of these interfaces. We demonstrate this in an experiment (see Figure 1) where we equipped a single node with two radio interfaces configured to non-interfering channels 1 and 11.

We ran two experiments — a two interface case, where both interfaces were active, and a single interface case, where one of the interfaces was disabled. In each experiment, the active interface(s) performed a "ping flood", where ICMP *Echo Request* packets were transmitted about every 10 ms. The average ping round-trip time (RTT) was around 1.7 ms. In Figure 1 we plot the tail of the RTT distribution, i.e., the round-trip times for packets with RTT greater than 5 ms. It is instructive to see that the packets in the two-interface simultaneous transfer experiment experienced higher interference, despite being on non-overlapping channels, as illustrated by the significantly higher number of pings with high RTTs (154 packets for two simultaneous interfaces versus 10 for single interface).

Based on these observations, the design of MultiScan makes use of one radio interface as the primary data transfer interface, while the other (secondary) interface is used to facilitate a fast 'make-before-break' handoff as and when necessary, for example, if the performance of the primary interface is deteriorating.

In the rest of this paper we will present design details of MultiScan and a detailed evaluation of the multi-radio handoff approach, as conducted in our wireless testbed. In particular, we will look at how it helps to eliminate handoff latencies and thus improve performance for VoIP applications like Skype.

2 Background

A typical WLAN consists of a number of APs. In order to reduce interference, neighboring APs operate on independent (non-interfering) channels. Different 802.11 standards have different number of such channels available, for example, in the US, IEEE 802.11b operates on 3 such channels: 1, 6, 11. A client moving from the coverage area of one AP to another in a WLAN needs to change its association accordingly in order to stay connected. The ensuing handoff process consists of the following stages:

1. Scanning: In this stage clients discover available APs by observing *beacon* frames that are periodically broadcast by APs. Scanning can be either passive, where a client simply waits for beacons, or active, where a client actively

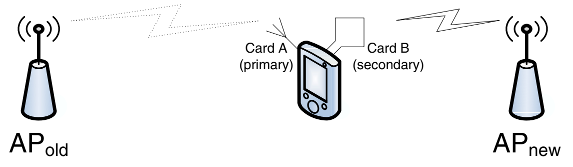


Figure 2: Multi-radio handoff scenario in MultiScan.

solicits beacons. An AP typically operates in one of many channels (e.g., 11 channels in 802.11b/g) and therefore a scanning client attempts to find APs in each channel.

2. Authentication: Access to 802.11 networks can be restricted by using such mechanisms as WPA. In these WLANs clients are required to exchange authentication messages with the AP identified for association.

3. Association: In this stage the client associates with the new AP by sending an *Association Request* and receiving an association ID. If *Inter Access Point Protocol* [12] is implemented, the new AP will inform the old AP that the client's association has changed, and obtain frames buffered at the old AP that are destined for the client.

In the scanning stage the interface has to switch between channels, and hence cannot be used for communication. Prior work has shown that over 90% of the time in the handoff process is spent in the scanning stage [7]. Because of this, work in optimizing handoffs has focused on making scanning more efficient [7, 2]. In the next section we show that such optimizations are not critical (though still useful) to network nodes with multiple network interfaces, since the second interface can be used to perform all association-related tasks with a new AP.

3 Handoffs in MultiScan

In the multi-radio scenario, we assume that a node has two interfaces: the *primary* interface and the *secondary* interface. Suppose that the primary interface is associated with AP_{old} and is used for communication, while the secondary interface is available to perform other tasks (see Figure 2). Clearly, such multi-radio node will have an advantage since it will be able to communicate normally and perform management operations simultaneously.

In a naive approach, the secondary interface could perform the scanning stage (which is the most time consuming stage of a handoff), while the primary interface is communicating normally with its AP. Once the secondary interface determines an AP to which the node needs to connect next, the primary interface could start the handoff process skipping the scanning stage. This optimized handoff can be performed in less than 5 ms. Besides the delay due to the last two stages of handoff, just switching the card to a different channel may require as much 20 ms [7], depending on chipset, which is significant for real-time applications. Although not the best we can do with multiple interfaces, this naive approach vastly reduces latency due to handoff and is absolutely safe, since from the AP infrastructure's point of view, the node does not do anything unexpected, it simply appears as if the node knows which AP to connect to without a scan.

In a more aggressive approach, we can eliminate handoff latency if the secondary interface proceeds to associate with AP_{new} while the primary interface is transfering data using AP_{old}. Once the secondary interface has finished its association process, the roles of the two interfaces are swapped, i.e., the secondary interface starts functioning as the primary interface and the previously primary interface dissociates with AP_{old} and starts operating as the secondary interface. This is our approach:

1. Normal operation: Communication is performed using the primary interface that is associated to AP_{old}, while the secondary interface is performing other tasks, possibly including scanning the channels.

2. Re-association: If it is determined that it would be beneficial to switch to a new AP, the second interface commences association with the new AP while the primary card is still used for data transfer with the old AP.

3. Interface Switch: As soon as the secondary interface is associated with the new AP, all of the node's outgoing traffic is sent via the secondary interface. The primary interface effectively becomes invisible, but stays up for some time to receive packets that may arrive delayed from AP_{old} because of buffering or a slow bridging tables update.

4. Completion: Primary and secondary interfaces switch roles: the formerly secondary interface becomes primary and is used for communication, and the formerly primary interface is freed to be used for other tasks.

Clearly, such approach potentially completely eliminates handoff latency (i.e. latency due to both the handoff process *and* switching the wireless interface to a different channel). Still, under certain conditions, connectivity during a MiltiScan handoff can be negatively impacted due to lost packets (though to a much lesser degree than during a single-interface vanilla handoff). Packets queued on the primary interface will be lost if AP_{old} learns that the node is associated with a different AP and will no longer accept node's packets. This is can happen if the channel of the primary interface is much more congested than the channel of the secondary interface.

Address management

An explicit goal of MultiScan is to require no changes in the APs or the wired infrastructure. In order to facilitate this goal, we require that both interfaces use the same IP and MAC addresses. Standard utilities (e.g., ifconfig) allow clients to set MAC addresses of individual interfaces as desired. Therefore, from the point of view of the infrastructure, a MultiScan handoff appears as if a single-radio wireless client just re-associated with a different AP (with zero

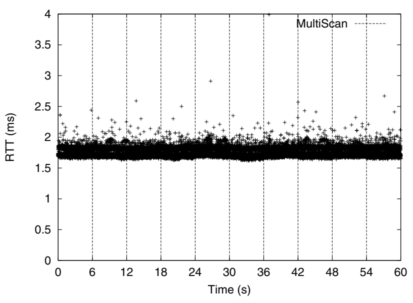


Figure 3: Ping RTTs with periodic handoffs for MultiScan client.

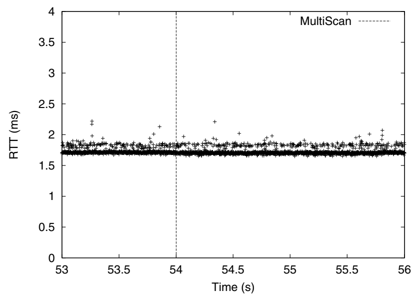


Figure 4: Zooming in on Figure 3 for a single handoff instant.

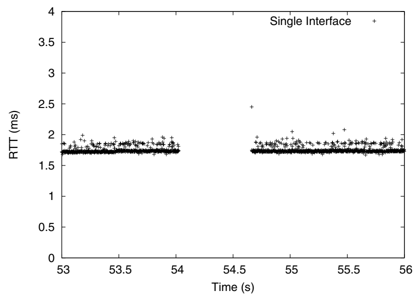


Figure 5: A handoff by a single interface client (see Figure 4).

latency). Note that when a handoff occurs, i.e., a secondary interface associates with a new AP, that AP automatically broadcasts a gratuitous ARP in the LAN announcing the new association. This updates cached ARP entries in different network devices.

4 Experimental Evaluation

In our evaluation of MultiScan, we used computers running Gentoo Linux 2005.0, equipped with two Atheros AR5212-based wireless cards, operating on non-overlapping 802.11b channels (one on channel 1, another on 11). Antennae were separated so as not to cause interference as in Figure 1. Two independent APs were set up on one of the hosts (each one using its own wireless interface). A single computer hosted both APs so that the same clock was used for all measurements. Data gathered in experiments where each AP ran on a separate computer were no different than data presented here. There were no 802.11 networks in the vicinity of the testbed.

A MultiScan module installed at the client controlled handoffs and interface switchting. MultiScan can trigger a handoff in a flexible manner, e.g., when the signal strength observed by the primary interface weakens. We performed both signal strength based handoffs as well as intentionally triggered handoffs — the latter gave us more control and allowed to stress-test the performance of MultiScan. `tcpdump` traces were obtained at both the APs and the client and were used to measure latency and other relevant parameters.

To test the performance of VoIP with MultiScan, we used a popular commercial software, Skype v1.0.0.20. In this section, in the interest of space we report only on some of the interesting results from our experiments that illustrate the key performance aspects of MultiScan.

4.1 ICMP ping floods and handoffs

In the first set of experiments we used ICMP ping flooding (in which *Echo Request* packets are continuously sent on the wireless link). The pings were sent from the wireless client to the AP(s), i.e., just across the wireless link and back. To stress-test MultiScan, we performed an experiment in which 10 handoffs were performed by the MultiScan module in a one-minute period. Figure 3 illustrates the typical RTTs of a stream of ping packets. The vertical lines in the figure indicate the times when the handoffs were initiated. From the figure it is apparent that ping traffic experienced no perceptible latency increase *due to MultiScan handoffs*. A careful observer will notice that the degree of density of data points in the figure varies, depending on which card is used. This is not an artifact of MultiScan, but rather, our hardware (one interface is actually slightly slower than the other, either due to hardware or heat issues).

To illustrate that the ping traffic experienced no perceptible handoff latency, we zoom in on one representative handoff instant in Figure 4. We next compare performance of MultiScan handoff to that of a typical vanilla single interface handoff in Figure 5. The figure shows a 640 ms outage period (x-axis range is same as in Figure 4). The rate of traffic in ping floods is fairly high, especially when the it goes across the wireless link only. Given the imperceptible change in performance for wireless handoffs when using MultiScan we feel confident that MultiScan will efficiently handle any traffic volume in the wireless link.

4.2 Skype and handoffs

The experiments in this section consisted of transmitting a one-minute audio file of a person talking through Skype. We first established the baseline performance over a wireless link without handoffs, and then compared the results with data obtained with MultiScan and vanilla handoffs. We have considered two metrics: end-to-end latencies and audio quality. However, the latency data was no different than the data presented in the previous section. This is not surprising since Skype's natural inter-packet latency (15 to 30 ms) was higher than that of ping floods (about 10 ms). Therefore, here we will focus only on audio quality.

Informal qualitative analysis revealed that MultiScan handoffs had no perceivable impact on audio quality, while single-card vanilla handoffs resulted in long periods (about half a second) of dead air. To quantify the differences in the audio we used cross-correlation of the captured samples.

Cross-correlation of two real functions $f(t)$ and $g(t)$ is

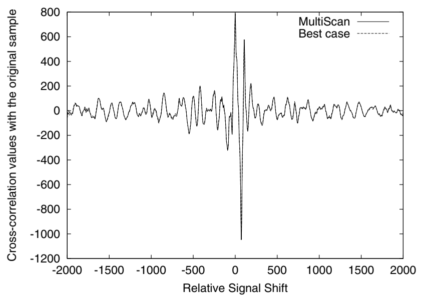


Figure 6: Cross-correlations of MultiScan and best case scenario.

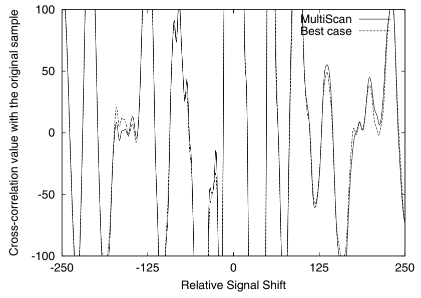


Figure 7: MultiScan and best case (Figure 6 close-up).

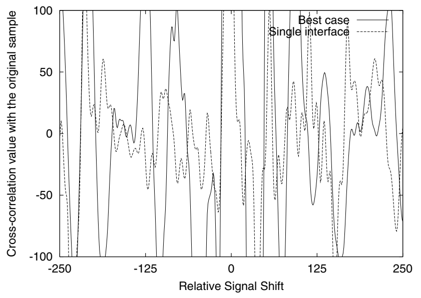


Figure 8: Cross-correlations of single-interface handoff and best case scenario.

defined as: $f \star g = \int f(\tau)g(t+\tau)d\tau$. The cross correlation function captures the similarity between the two functions. In particular, two signals that are similar should have a high cross-correlation near the origin and low cross-correlation elsewhere.

In Figures 6 through 8, a line represents the cross-correlation between the original audio signal (at the sender) and the received audio signal across the wireless link.

The best case in all of the experiments reflects the situation where no handoffs are performed and the audio signal is transmitted uninterrupted over the wireless link. Note that even in the best case, Skype performance is not perfect, just adequate for normal voice communication. In Figure 6 we show the cross-correlation function of two different scenarios — one in which there were 10 MultiScan handoffs over a minute interval, and another is the best case just described. The two plots are virtually indistinguishable from each other. The maxima of the two cross-correlations are 788.3870 and 737.2810 for the best case and MultiScan, respectively. These differences are very minor and are comparable to the differences between two independent best case (no handoffs) transmissions. Figure 7 is a close-up of Figure 6. The differences in cross-correlation values are small and the two samples sounded identical.

In contrast to Figure 7, consider Figure 8, where we show the performance difference between the best case (single interface, no handoffs) and a case where the client performed a *single* one-interface vanilla handoff. The two plots are significantly different and the loss in audio quality was perceivable to the human ear.

5 Related Work

Researchers have used multiple radios to improve performance in a number of different applications. Some examples include: reducing energy consumption of wireless clients, e.g., Wake-on-Wireless [4], improving web performance in wide-area (cellular) networks, e.g.,work by Rodriguez et al. [9], and in constructing wireless mesh networks, e.g., commercial ventures like MeshDynamics, and research efforts in Microsoft Research, Seattle [5] and Intel Research, Cambridge [10]. In particular, Bahl et al. [6] make an explicit case for multi-radio wireless systems for improved performance.

In this paper we take another step in advocating multi-radio wireless node design and demonstrate its applicability in WLANs to improve VoIP application performance. To the best of our knowledge, there has been no work in eliminating handoff latency in WLANs using multiple radios and demonstrating advantages of such technique for VoIP applications.

Prior research has focused on improving handoff performance using a single radio interface. Shin et al. [3] in the Neighbor Graphs work explore techniques to improve handoffs by implementing a topology inferencing technique in both clients and APs. Ramani et al. [2] defined a technique called SyncScan that requires appropriate time synchronization between APs and clients. SyncScan also requires synchronization of *Beacon* broadcast times for different APs and periodic channel hopping of clients. Both schemes seek to reduce the time spent in the channel scanning phase when a handoff occurs. By changing the APs and the clients, and by increasing coordination between them, Neighbor Graphs achieves handoff latency of about 40 ms, and SyncScan handoffs take 2-3 ms (but the technique requires periodic suspension of communication that could last more than 10 ms, depending on hardware).

Unlike the above schemes that attempt to optimize performance with a single radio but require coordination and cooperation between APs and clients, MultiScan relies only on multiple radios in wireless clients to completely eliminate handoff latencies. Our proposed scheme requires no interaction or participation from APs, and hence can be deployed in arbitrary wireless environments, including environments where neighboring APs are not administered or controlled by a single entity. Such scenarios are becoming commonplace in many major cities around the world in the form of community wireless networks [14]. Table 1 summarizes the differences between the approaches discussed above and MultiScan.

Two other works are related to our efforts insofar as they apply to multi-interface nodes in general. Adya et al. [8] defined a protocol called MUP, which allows multi-radio

	Wireless interfaces	Handoff latency	Infrastructure modification
Neighbor Graphs	1	$\sim$ 40 ms	yes
SyncScan	1	2-3* ms	yes
MultiScan	2	0 ms	no

* SyncScan requires routine suspension of communication that could last for more than 10 ms, depending on hardware.

Table 1: Comparison of different handoff mechanisms.

wireless nodes in a mesh network to potentially establish two separate wireless links between a pair of nodes. However, the authors advocate the use of only one of these links at a given time based on channel conditions. This work was primarily focused on improving efficiency of wireless mesh networks.

Finally, work by Chandra et al. [13] demonstrated how a node could stay connected to multiple wireless networks simultaneously. Their approach is based on having the radio interface change channels unbeknownst to the applications. In the context of MultiScan, this work applies to the potential functionality of the secondary interface.

6 Conclusion

It is not surprising that network nodes with multiple network interfaces can experience better performance than nodes with a single network interface. Many hard-to-overcome limitations of 802.11 wireless networking (such as short communication range, vulnerability to environmental noise, and relatively low throughput in many practical scenarios), coupled with the ever-increasing demand from the application side for bandwidth and low latency make it natural that multi-interface options be explored. While adding a radio interface leads to a modest increase in cost, this and other works demonstrate that significant performance improvements can be achieved. Therefore, we would like to reinforce the need for increased availability of multi-radio interfaces in wireless devices.

Overall, we make the following observations and contributions in this work:

- We recommend the use of two radio interfaces in eliminating handoff latencies in WLANs. Using two radio interfaces in wireless devices is already feasible and will be more so with the increased availability of multi-interface and multi-band wireless cards.
- Our multi-radio approach does not use the radios in tandem for data transfer, as ensuing interference between the interfaces themselves (even when they are on independent wireless channels) can lead to degraded performance. Instead, one of the interfaces should be used as the primary data interface while the other serves as a secondary interface (periodically) monitoring the environment for handoff opportunities. The functionality of the two interfaces are swapped when necessary.
- Utilization of multiple radios does not create any additional load on wireless spectrum resources. This is because at any time one wireless interface acts as the secondary and does not impose any data load on the wireless medium. This also implies that the proposed mechanism is not hindered as more clients start operating in the multi-radio mode.
- We have developed MultiScan as an open source Linux module that will shortly be available for public downloads from: http://www.cs.wisc.edu/$\sim$suman/projects/multiscan/.

As a followup to this work, we intend to explore how MultiScan should be extended to handle the newly defined Inter Access Point Protocol (IAPP) [12]. IAPP is a new mechanism proposed by the IEEE 802.11f working group to better handle roaming clients (currently IAPP is not widely implemented or available). IAPP, among other requirements, enforces unique AP association and hence timing of AP switch operation currently implemented in MultiScan needs to be appropriately optimized.

References

[1] IEEE, 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications 1999.

[2] I. Ramani and S. Savage, "SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks" *Proceedings of the IEEE Infocom, March 2005.*

[3] M. Shin, A. Mishra and W.A. Arbaugh, "Improving the Latency of 802.11 Hand-offs using Neighbor Graphs" *Mobisys 2004 June, 2004, Boston, USA.*

[4] E. Shih, P. Bahl, and M. Sinclair "Wake on wireless: an event driven energy saving strategy for battery operated devices" *ACM Mobicom, Oct 2002.*

[5] R. Draves, J. Padhye, and B. Zill "Routing in multi-radio, multi-hop wireless mesh networks" *ACM Mobicom, Oct 2004.*

[6] P. Bahl, A. Adya, J. Padhye, A. Walman "Reconsidering wireless systems with multiple radios" *ACM Sigcomm Computer Communications Review, Vol. 34, No. 5, Oct 2004.*

[7] A. Mishra, M. Shin, and W. Arbaugh, "An Empirical Analysis of the IEEE 802.11 MAC layer Handoff Process" *ACM Computer Communications Review, vol. 33, no. 2, Apr. 2003.*

[8] A. Adya, P. Bahl, J. Padhye, A. Wolman, L. Zhou "A Multi-Radio Unification Protocol for IEEE 802.11Wireless Networks" *BROADNETS 2004, San Jose, CA, October 2004*

[9] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee "MAR: A Commuter Router Infrastructure for the Mobile Internet" *ACM Mobisys*, June 2004.

[10] J. Robinson, K. Papagiannaki, C. Diot, X. Guo, and L. Krishnamurthy "Experimenting with a Multi-Radio Mesh Networking Testbed" *Workshop on Wireless Network Measurements (WiNMee), April, 2005.*

[11] F. K. Al-Bin-Ali, P. Boddupalli, and N. Davies, "An Inter-Access Point Handoff mechanism for Wireless Network Management: The Sabino System" *in Proceedings of the International Conference on Wireless Networks , Las Vegas, NV, June 2003*

[12] IEEE, 802.11f: IEEE Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation *IEEE Standard 802.11f, 2003.*

[13] R. Chandra, P. Bahl, and P. Bahl. MultiNet: "Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card" *IEEE Infocom 2004.*

[14] Public Wireless Community Network List http://www.toaster.net/wireless/community.html

[15] Skype Technologies, http://skype.com/

Estimation of Link Interference in Static Multi-hop Wireless Networks

Jitendra Padhye[†], Sharad Agarwal[†], Venkata N. Padmanabhan[†], Lili Qiu[‡], Ananth Rao[§], Brian Zill[†]

[†]Microsoft Research [‡]University of Texas, Austin [§]University of California, Berkeley

We present a measurement-based study of interference among links in a static, IEEE 802.11, multi-hop wireless network. Interference is a key cause of performance degradation in such networks. To improve, or to even estimate the performance of these networks, one must have some knowledge of which links in the network interfere with one another, and to what extent. However, the problem of estimating the interference among links of a multi-hop wireless network is a challenging one. Accurate modeling of radio signal propagation is difficult since many environment and hardware-specific factors must be considered. Empirically testing every group of links is not practical: a network with n nodes can have $O(n^2)$ links, and even if we consider only pairwise interference, we may have to potentially test $O(n^4)$ pairs. Given these difficulties, much of the previous work on wireless networks has assumed that information about interference in the network is either known, or that it can be approximated using simple heuristics. We test these heuristics in our testbed and find them to be inaccurate. We then propose a simple, empirical estimation methodology that can predict pairwise interference using only $O(n^2)$ measurements. Our methodology is applicable to any wireless network that uses omni-directional antennas. The predictions made by our methodology match well with the observed pairwise interference among links in our 22 node, 802.11-based testbed.

1 Introduction

Multi-hop wireless networks have been a subject of much study. Most of the original work in this area was motivated by scenarios in which the nodes were highly mobile. Recently, interesting commercial applications of *static* multi-hop wireless networks have emerged. One example of such applications is "community wireless networks" [2, 11]. Several companies [9, 14] are field-testing wireless "mesh" networks to provide broadband Internet access.

Interference among wireless links significantly impacts the performance of static multi-hop wireless networks. Several researchers have studied this issue. The impact of interference on the capacity of wireless networks is studied in [8, 10, 12], while the impact on the performance of transport-level protocols is considered in [6, 7, 13]. The need for routing protocols to take link interference into account has been underscored in [4, 5]. Information about link interference is also needed for optimal channel assignment [16].

Many of these studies build upon the knowledge of which links in the network interfere with each other. Yet, the problem of *estimating* the interference among links in a multi-hop wireless network has not been adequately addressed.

The problem of estimating link interference can be described informally as follows: given a set of wireless links, estimate whether (and by how much) their aggerate throughput will decrease when the links are all active simultaneously, compared to when they are active individually.

This is a challenging problem for several reasons. Accurate modeling of radio signal propagation is difficult since many environment and hardware-specific factors must be considered. Empirically testing every group of links for interference is not practical: a network with n nodes can have $O(n^2)$ links, and even if we consider only pairwise interference, we may potentially have to test $O(n^4)$ pairs. The interference pattern could change due to environmental factors, so interference estimation is not a one-time task. Hence, it is important to do it efficiently.

Given these difficulties, some researchers simply assume that the information about which links in the network interfere with each other is known apriori [10]. Others assume that it can be approximated by using simple heuristics. One common heuristic states that the interference range equals a small multiple (typically, a factor of 2) of the communication distance [8, 18].

In this paper, we study the phenomenon of interference among links in a 22-node IEEE 802.11 a/b/g based indoor wireless testbed. The paper makes two contributions. First, we show that the simple heuristics described in the previous literature fail to accurately predict the interference among links in our testbed. Second, we propose a simple, empirical estimation methodology to predict pairwise interference that requires only $O(n^2)$ measurement experiments. We show that the predictions made by our methodology match well with the observed pairwise interference among links in our testbed under a variety of conditions. Our methodology is useful for any wireless network where nodes use omni-directional antennas. We focus on omni-directional antennas since these are cheap and easy to deploy, and hence popular. Network architectures based on omni-directional antennas are quite common [17].

Paper outline: First, we formalize the notion of pairwise interference among wireless links. Next, we present a brief description of our testbed. We show that the simple heuristics used in previous work do not accurately model the interference in our testbed network. We next present our empirical methodology, and show that it accurately predicts pairwise link interference in our testbed. Finally, we summarize related work and present our conclusions.

2 Interference among wireless links

In this section, we define a metric to measure interference between a pair of wireless links. We assume that nodes communicate using the IEEE 802.11 protocol; parameters such as transmit power, data rate etc. are all set to fixed values; and the background noise level is constant. We also assume that RTS/CTS handshake is disabled for all nodes, which is the default behavior for most wireless cards.

We start by defining what constitutes a wireless link. Unlike in a wired network, the links in a wireless network are not well-defined. For the purposes of this paper, we define wireless links using packet loss rate. We say that a link from node A to node B, denoted by L_{AB}, exists if the packet loss rate in either direction does not exceed some threshold. We defer a detailed discussion of the definition until Section 4.2.

We now define a metric to measure interference between a pair of links. Consider links L_{AB} and L_{CD}. For some fixed packet size, let U_{AB} denote the unicast throughput of the link L_{AB}, when no other links are active in the network. Similarly define U_{CD} for link L_{CD}. Now assume that both L_{AB} and L_{CD} are active simultaneously. Let their respective unicast throughput be denoted by $U_{AB}^{AB,CD}$ and $U_{CD}^{AB,CD}$. Define the *link interference ratio* as:

$$LIR_{AB,CD} = \frac{U_{AB}^{AB,CD} + U_{CD}^{AB,CD}}{U_{AB} + U_{CD}} \quad (1)$$

Thus, LIR is the ratio of aggregate throughput of the links when they are active simultaneously, to their aggregate throughput when they active individually.

LIR takes values between 1 and 0. The maximum value of LIR is 1, which means that the aggregate throughput does not decrease when the links are active simultaneously. *Thus, $LIR = 1$ implies that the links do not interfere.* A value of LIR less than 1 means that the aggregate throughput of the links decreases when they operate simultaneously. *Thus, $LIR < 1$ implies that the links interfere with each other*. The links can interfere with each other due to several reasons, listed below. Consider two links, L_{AB} and L_{CD}:

Carrier Sense: The 802.11 protocol requires the sender to monitor the radio channel for signs of activity, prior to transmitting a packet. If any activity is detected, transmission is deferred until a later time [1]. This is known as *carrier sensing*. If the two senders, A and C are within the carrier sense range of each other, then only one of them will transmit at a time. Otherwise, they may both transmit, and one of the following may occur.

Data-Data Collision: The transmission by C may generate sufficient noise at B to interfere with reception of the packet being sent by A. A similar "collision" may occur at D. This is known as the hidden terminal problem.

Data-ACK Collision: For unicast communication, the 802.11 protocol requires the receiver of a packet to transmit an acknowledgment to the sender. If node D successfully receives the data packet sent by C, it will transmit an ACK. This transmission may interfere with ongoing reception of data packet at B. A similar collision may occur at D.

ACK-Data Collision: The data packet sent by C may interfere with ongoing reception of ACK sent by B at A. A similar collision may occur at C.

ACK-ACK Collision: The ACK sent by D may interfere with the reception of ACK sent by B at A. A similar collision may occur at C.

[1]This is a simplified description of the actual protocol.

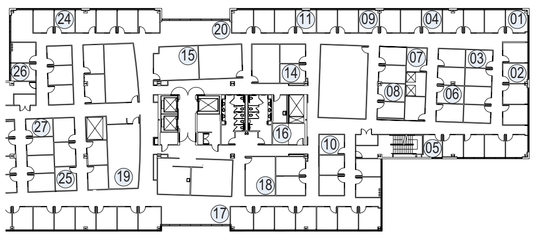


Figure 1: Layout of our testbed

A typical value of LIR is 0.5, which means that the aggregate throughput of the links is halved when they are active together. This usually (but not always) happens when the senders are within carrier sense range of each other. The minimum value of LIR is 0. This means that the links get zero throughput when they operate together. This can happen if the senders are not within the carrier sense range of each other, and collisions at the receiver are frequent.

In practice, we see a range of LIR values instead of just the three described above. They can result from packet losses, variable nature of background noise etc. Many of the simple heuristics used to estimate link interference only predict whether a pair of links interfere with each other or not. In other words, they only predict LIR is less than 1. We term this as the "binary" notion of interference.

3 Testbed

The experimental data reported in this paper was collected using a 22-node wireless testbed, located on one floor of a typical office building. The nodes are placed in offices, conference rooms and labs (Figure 1). All rooms have floor-to-ceiling walls and wooden doors. The nodes were not moved during testing. Each node is equipped with two 802.11 wireless cards: a Proxim ORiNOCO a/b/g Combo-Card Gold, and a NetGear a/b/g WAG 511. For experiments described in this paper, the two cards were never active simultaneously. RTS/CTS handshake was disabled (by default). All cards operated in the 802.11 ad-hoc mode. We used the built-in antennas of these cards, which are roughly omni-directional.

4 Performance of Simple Heuristics

In this section, we consider the performance of three simple heuristics from previous literature. The experiments described in this section use the following settings. All Orinoco cards were turned off. All Netgear cards were set to operate in 802.11a mode, on channel 36, at full transmit power. The transmit rate of each card was fixed at 6Mbps. We use 802.11a mode for these tests since our building has an operational 802.11b network.

4.1 The Heuristics

The first heuristic that we consider is used in [4, 5]. It assumes that all links on a multi-hop wireless path interfere with each other. In a connected network, we can always construct a path that includes a given pair of links. In short, this heuristic assumes that any two links in the network interfere with each other. This is clearly a pessimistic model.

The second heuristic [12] assumes that two links in the network interfere only if they share an endpoint. This is an optimistic heuristic. It is commonly known as the *point interference* model.

We do not expect either of these two heuristics to work well in our testbed. We include these in our study because they represent the two extreme ends of the approximations that have been used in the literature. Our measurements indeed show that these heuristics perform poorly.

The third heuristic [8, 18] is more sophisticated. Consider two links L_{AB} and L_{CD}. Let d_{AB} be the distance between nodes A and B. Similarly define d_{CD}, d_{BC} and d_{AD}. The model says that L_{AB} and L_{CD} will interfere with each other if either $d_{BC} \leq K * d_{AB}$ or $d_{AD} \leq K * d_{CD}$. A commonly used value for K is 2. Intuitively, the model says that if a node is receiving a transmission, a second transmitter can interfere with that reception only if it is sufficiently close. This model is generally paraphrased as *interference range is twice the communication distance*.

We term these three heuristics as $M1$, $M2$ and $M3$, respectively. These heuristics are "binary"models, since they only predict whether a given pair links interfere with each other or not. They do not predict the actual LIR.

To see how these three models perform in our testbed, we compare their predictions for several pairs of links against experimentally measured LIR. The first step in this process is to select a set of links in the testbed to experiment with.

4.2 Which links to use?

We define wireless links using packet loss rate as in [4]. Packet loss rates are easy to measure and reflect the link quality experienced by higher layers. For a pair of nodes that communicates using the 802.11 protocol, packet loss rate in both directions matters, since a unicast packet transmission is considered successful only if the sender successfully receives the ACK sent by the receiver. To discover links that have reasonably low packet loss rate in both directions, we carried out the following experiment.

We had each node in our testbed broadcast 1000 byte packets for 30 seconds. Only one node was active at a time. We measured the packet reception rate at all other nodes in the network. The entire test was repeated 50 times. This data gives us the average packet loss rate between every ordered pair of nodes in our testbed. For two nodes A and B, let P_{AB} be the packet loss rate from A to B, and let P_{BA} be the loss rate from B to A. We say that links L_{AB} and L_{BA} exist if: $1/((1 - P_{AB}) * (1 - P_{BA})) \leq \alpha$, where $\alpha \geq 1$ is some threshold value. This definition was proposed in [4], where the ratio is called the ETX (expected transmissions) value of the link. For the purposes of our paper it is sufficient to note that a high ETX value implies that the link is lossy in either one or both directions.

We use the threshold of $\alpha = 3$ in the rest of the paper, to weed out highly lossy links. Any reasonable routing protocol will avoid such poor quality links. Of the $22 * 21 = 462$ possible links in our testbed, 152 links have $\alpha \leq 3$. The average loss rate of these 152 links is 2.9%. We have ex-

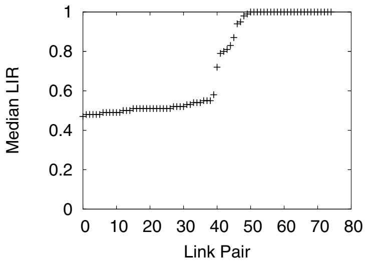


Figure 2: Median LIR of 75 link pairs

perimented with lower values of α; lower values reduce the number of links, but our interference results remain similar.

Note that we measure packet loss rate in both directions with 1000 byte packets, even though ACK packets are much smaller than data packets. It is not our aim to accurately characterize the loss rate of a link - we only want to eliminate highly lossy links that a routing protocol will avoid. Similar approach has been used in [4, 5].

4.3 Estimation of interference

Given the 152 links as defined above, there are a total of 11476 possible link pairs in our network. We ignore pairs in which the links share at least one endpoint, since such links will always interfere with one another. After removing such pairs, we are left with 9168 link pairs, which are still too many to be tested exhaustively. In this paper, we present results for 75 of these pairs, selected at random. We have also done some experiments with larger groups of link pairs, and have seen similar results.

For each selected link pair, we measured LIR as follows. For each link in the pair, we measured the unicast throughput using 1000 byte UDP packets for 30 seconds. Immediately afterwards, we measured the aggregate throughput of the two links operating together, again using unicast UDP packets for 30 seconds. Using the definition in Equation (1) we calculated the LIR for this pair. Testing links in a pair in quick succession helps mitigate the impact of environmental variations. We repeated the experiment 5 times for each of our 75 link pairs. Thus the total duration of the experiment was just under 10 hours The median LIR value for these 75 link pairs are shown in Figure 2. Note that testing all 9168 pairs would have required more than 1100 hours.

First, note that we have several link pairs with intermediate LIR values between 0.5 and 1. In other words, interference is not a binary phenomenon. To compare this data with the binary predictions of $M1$, $M2$ and $M3$ models, we must pick a threshold, β. If $LIR < \beta$, we deem the links to have interfered in our experiment. If $LIR \geq \beta$, we deem that the links did not interfere.

Of the 75 node pairs, 24 pairs have LIR of 1. Thus, in each of these 24 pairs, the two links do not interfere with each other. Five other link pairs have LIR values between 1 and 0.9. Given the minimal interference, we classify these link pairs as non-interfering as well. Thus, we set $\beta = 0.9$. With this threshold, we have 29 pairs in which links do not interfere, and 46 pairs in which the links do interfere.

We see that the $M1$ model is too pessimistic for our net-

	M3 Prediction	
	Interference	No interference
Observed Interference	46	0
Observed No Interference	10	19

Table 1: Performance of M3 model

work, since we do have 29 non-interfering link pairs. On the other hand, the $M2$ model is too optimistic. The two links in each pair do not share an endpoint, so according to the $M2$ mode, none of the link pairs should show any interference. Yet, we have 46 link pairs in which the links *do* interfere.

The $M3$ model is harder to verify. It is defined in terms of distance between nodes. We found that the predictions made using distance are quite inaccurate in our testbed. In an indoor testbed like ours, the radio signal propagation is also affected by office walls and other obstacles. There is no easy way to incorporate this information in the model. Therefore, we define a variant of the $M3$ model that does not rely on physical distance between nodes. We will say that a pair of links L_{AB} and L_{CD} interfere if there is a 2 hop (or shorter) path from C to B, or from A to D. In other words, the modified model says that a pair of links will interfere if the sender of one link is within two hops of the other link's receiver. Note that "hop" is just another term for a wireless link. This variant of the $M3$ model predicted that 56 of the 75 link pairs will show interference. In our experiments, we observed interference in only 46 of these 56 pairs. The other 10 pairs did not show interference in our experiments. On the other hand, the model predicted no interference for 19 pairs. We indeed did not observe interference in any of these 19 pairs. These numbers are summarized in Table 1. The conclusion is that the model is pessimistic: it errs on the side of predicting interference even when there is none.

It may appear that the model seems pessimistic because we used $\beta = 0.9$ to classify experimental observations, and it is too low a threshold. However, even if we use $\beta = 1$ to classify experimental observations (and hence classify more pairs as interfering), the model still incorrectly predicts interference in 7 pairs that do not see any interference.

The pessimistic nature of the model is probably due to the indoor setting of our testbed. In such an environment, the radio signal degrades much faster than it would in free space, thus limiting the overall interference. We also evaluated a 1-hop variant of the model, which turned out to be optimistic. We believe that it may be possible to modify the 1-hop variant further to provide better predictions. However, there is no guarantee that the predictions of the 1-hop model will be accurate in other environments. Furthermore, even the improved model will provide only binary predictions. In the following section, we present a measurement-based approach which automatically takes into account the impact of environmental factors, and is capable of predicting intermediate values of LIR.

5 Proposed empirical methodology

In the previous section, we showed that the simple models proposed in the literature do not accurately predict the interference in our testbed. We now present a simple empirical methodology to estimate LIR.

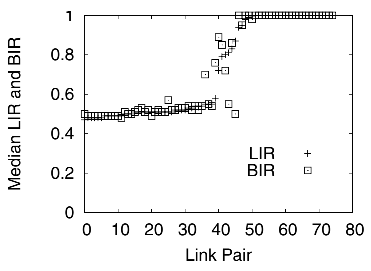


Figure 3: Median LIR and BIR of 75 pairs.

Recall that in Section 2, we listed several reasons why two links may impact each other's throughput. If we ignore the ACKs (given their relatively small size), we can then use the following simple methodology to estimate the impact of carrier sensing and collision of data packets as follows.

First, have one node, say A, broadcast packets as fast as it can. Only one node is active at a time. Denote the send rate by S_A. Keep track of the delivery rate of packets at all other nodes in the network. For example, the delivery rate at node B will be denoted by R_{AB}. Have each node broadcast in turn. Then, select a pair of nodes, say A and C, have them broadcast packets together. Denote their send rates by S_A^{AC} and S_C^{AC}. At all remaining nodes measure the delivery rate of packets they receive from each of the two broadcasting nodes. For example, at node B, the delivery rate of packets from A is denoted by R_{AB}^{AC}. Similarly, at node D, the delivery rate of packets from C is denoted by R_{CD}^{AC}. Have each pair broadcast in turn. Thus, we have carried out a total of $O(n^2)$ experiments.

Consider links L_{AB} and L_{CD}. Using the data gathered from the above methodology, we can define the "broadcast interference ratio" (BIR) as follows.

$$BIR = (R_{AB}^{AC} + R_{CD}^{AC})/(R_{AB} + R_{CD}) \qquad (2)$$

Our hypothesis is that the BIR is a good approximation of LIR. If the hypothesis is true, we can estimate interference for every pair of links using only $O(n^2)$ experiments, while testing each link pair will require far more (potentially $O(n^4)$) experiments. This is a substantial improvement: if we use 30 second transfers, and repeat each experiment 5 times, calculating BIR for every pair of links requires just over 28 hours. The key idea is that we can estimate unicast interference using broadcast packets, if we ignore impact of ACKs.

There are several reasons to believe that our hypothesis is correct. It is easy to see that BIR captures impact of carrier sensing on the two senders. It also captures the impact of data packet collisions at the receivers. The ACK packets are quite small (only 14 bytes) and the chance of them colliding with each other is also small. There are also some reasons to believe that the hypothesis is not correct. First, if a broadcast packet is lost (say due to collision), it is not retransmitted. On the other hand, a lost unicast packet is retransmitted multiple times, and the mean wait time (802.1 backoff) before sending the next retransmission is doubled.

Thus, a lost unicast packet has a higher impact on throughput measured at the user level. Second, while ACK packets may not collide with one another, data and ACK packets can still collide. We now test our hypothesis experimentally.

5.1 Evaluation: Baseline scenario

To test the hypothesis, we performed the following experiment. We use the same settings (802.11a, full transmit power, transmission rate fixed at 6Mbps) that we used in Section 4 and consider the same 75 link pairs as shown in Figure 2. To minimize the impact of environmental factors, the broadcast experiments designed to measure BIR were performed just before the unicast experiments designed to measure LIR. The median values of BIR and LIR for each link pair are shown in Figure 3. We see that BIR matches LIR well in most cases. The CDF of the absolute error ($|LIR - BIR|$) is shown in Figure 4. The median of absolute error is zero, and the mean is 0.026. Given that $|LIR - BIR|$ can range from 0 to 1, the mean and the median are quite low. Thus, our methodology works quite well in this scenario.

These results bring up several interesting questions. First, does the methodology work for other scenarios? Second, note that we carried out the broadcast and the unicast experiments back-to-back. In reality, we must do all the broadcast experiments together, and then use the results to predict link interference. The question then becomes: if we do broadcast experiments separately, will BIR obtained at some point in time still match LIR observed at some later point? Third, is the model capable of telling us *why* two links interfere? We discuss these questions next.

5.2 Other scenarios

We considered three other scenarios to evaluate our methodology. In the first scenario, we turned on the autorate feature for each card. When the autorate algorithm is on, the transmission rate for unicast packets may vary over time, in response to changing noise levels etc. The rate selection algorithm is not standardized. The broadcast packets, however, are always sent at the lowest data rate (6Mbps for 802.11a). Note that in the baseline scenario, the unicast transmission rate was also fixed at 6Mbps. With autorate on, we would expect more mismatch between BIR and LIR.

In the second scenario, we reduced the transmit power on each card to 50% of the full power. We fixed the transmission rate at 6Mbps. At 50% transmit power, the network has fewer links: only 128, instead of 152. Thus, the link pairs used in this scenario are different from the link pairs used in the previous, full-power scenarios. The average loss rate of these 128 links is 4.6%, while the average loss rate at full power was 2.9%. Since, the links are more lossy in this scenario, we would expect slightly higher mismatch between BIR and LIR in this setting.

All the experiments so far were done in 802.11a mode, using the NetGear cards. In the third scenario, we turned off the Netgear cards, and used Orinoco cards, set to operate in 802.11g mode (i.e. in 2.4GHz spectrum), at full power, with rate fixed at 1Mbps (i.e. the lowest data rate for 802.11g). We have an infrastructure mode 802.11b network in our building, which operates in the same frequency band. We tested this scenario at night, to minimize the impact of interference from the WLAN, however, we would still expect to see higher error in this scenario.

For each of these scenarios, we measured BIR and LIR of 75 link pairs, using back-to-back experiments as before. The CDF of absolute error in each of the three cases is shown in Figure 5. The results show that our methodology performs generally well in each scenario. As expected, the mismatch is somewhat high for the autorate scenario. In the other two cases, the median error is only 0.01. Even in autorate case the median error is only 0.03, while the mean is 0.065. These three experiments increase our confidence in the general applicability of our method.

5.3 BIR and LIR measured 5 days apart

We used the same settings as the baseline scenario, but did only the broadcast experiments. We compare BIR calculated from these experiments with the LIR measured in the baseline experiment. The two experiments were done 5 days apart. The CDF of absolute error is shown in Figure 6. The graph also shows the baseline error CDF (labeled "Back-to-back"') for comparison purposes. We see that BIR is still generally a good predictor of LIR, but as expected, the error is somewhat higher compared to the baseline (back-to-back) case. The median error is only 0.01, and mean is 0.049. The results show that even in a static environment like ours, the interference patterns are slightly different at different times. The need to repeat interference measurements underscores the need for an inexpensive experimental methodology to measure interference.

5.4 Why do links interfere?

Our methodology also helps determine *why* two links interfere with one another. Consider links L_{AB} and L_{CD} that interfere with one another. During the broadcast experiments done to determine BIR, we had nodes A and C broadcast alone, as well as together. Consider the ratio of their *send rates*, when they were broadcasting together to when they were broadcasting alone. Define *carrier sense ratio:*

$$CSR = (S_A^{AC} + S_C^{AC})/(S_A + S_C). \tag{3}$$

Note that we are using broadcast packets, so both senders send at the same data rate. If two senders are within the carrier sense range of each other, then only one of them would be able to send at a time, resulting in a CSR value of 0.5. If the senders are not within each other's carrier sense range, CSR will be 1. Intermediate values can result from noise, differences in sensitivity of antennas, signal strength fluctuations due to environmental factors etc.

In the baseline scenario shown in Figure 2, 46 link pairs have $LIR < 0.9$, indicating some degree of interference. Of these, 34 link pairs had a CSR of 0.5. Thus, carrier sensing seems to be the major cause of interference in our testbed. We see similar results for the other three scenarios considered in Section 5.2. We believe that this is one of the reasons why BIR and LIR show a good match under all scenarios. We are currently investigating this issue further.

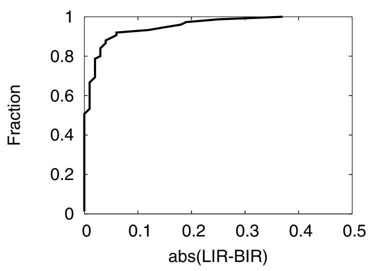


Figure 4: Baseline Scenario

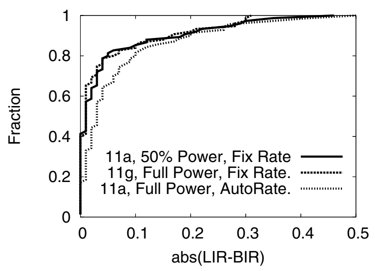


Figure 5: Three other scenarios.

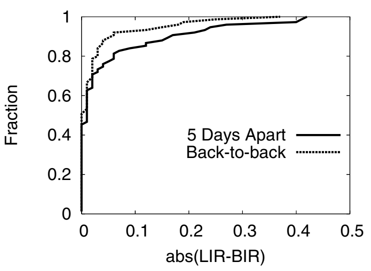


Figure 6: Measured 5 days apart

6 Related Work

The importance of studying wireless interference has long been recognized. For example, in [15] the impact of interference on fairness is considered. In [19] it is shown that paths with high degree of interference suffer disproportionately. Several researchers [8, 10, 12, 13] have considered impact of interference on the overall capacity of a multi-hop wireless network. However, each of these papers assumes that the information about link interference is available, but they do not describe how to estimate it. Thus our work on a practical method for estimating link interference helps generate the information taken for granted in previous work.

As we discussed earlier, various heuristics for estimating link interference have been proposed [4, 8, 12, 18]. We have shown that our empirical methodology can provide a more accurate estimation of pairwise link interference.

The knowledge of which links interfere with one another can benefit a number of network operations. For example, it can improve routing algorithms [4, 5], help in engineering and managing multi-hop wireless networks [3], and aid in channel assignment [16].

There is a large body of work on measuring various properties of wireless networks. Here we list some of the recent work. Aguayo et al. [1] analyze the causes of packet loss in an outdoor multihop 802.11b network. Yarvis et al [20] use testbeds in three different houses to study the properties of home wireless networks. Gupta et al [7] experimentally study the performance of TCP in a multi-hop wireless network. Our work contributes a practical technique for measuring another key property, viz., wireless interference.

7 Conclusion and Ongoing Work

In this paper, we considered the problem of estimating pairwise interference among links in a multi-hop wireless testbed. Using experiments done in a 22-node, 802.11-based testbed, we showed that some of the previously-proposed heuristics for predicting pairwise interference are inaccurate. We then proposed a simple, empirical methodology to estimate pairwise interference using only $O(n^2)$ measurements. We showed that our methodology accurately predicts pairwise interference among links in our testbed in a variety of settings. Our methodology is applicable to any 802.11-based wireless network where nodes use omni-directional antennas.

There are several avenues for future work. We hope to increase the accuracy of our methodology by accounting for the impact of four factors that we ignored in this paper. These four factors are: (i) retransmissions of lost unicast packets, (ii) RTS/CTS handshake (iii) collisions between data and ACK packets (iv) autorate algorithms.

We would like to extend our approach to estimate interference among larger groups of links, instead of just pairwise interference.

Finally, we note that our methodology requires nodes to generate broadcast traffic, and existing traffic on the network can significantly reduce the accuracy of our approach. We are currently exploring the possibility of determining interference patterns by simply observing correlation between existing traffic flows on the network.

References

[1] D. Aguago, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, 2004.
[2] Bay area wireless users group. http://www.bawug.org/.
[3] R. Chandra, L. Qiu, K. Jain, and M. Mahdian. On the placement of internet taps in wireless neighborhood networks. In *ICNP*, 2004.
[4] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. High-throughput path metric for multi-hop wireless routing. In *MOBICOM*, 2003.
[5] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh network. In *MOBICOM*, 2004.
[6] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The Impact of Multihop Wireless Channel on TCP Throughput and Loss. In *INFOCOM*, 2003.
[7] A. Gupta, I. Wormsbecker, and C. Williamson. Experimental evaluation of TCP performance in multi-hop wireless ad hoc networks. In *MASCOTS*, 2004.
[8] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Trans on Info Theory*, Mar 2000.
[9] Invisible Networks. http://www.invisible.uk.net/how/.
[10] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu. The impact of interference on multi-hop wireless network performance. In *MOBICOM*, 2003.
[11] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. In *HotNets*, 2003.
[12] M. Kodialam and T. Nandagopal. Charaterizing achievable rates in multi-hop wireless newtorks: The joint routing and scheduling problem. In *MOBICOM*, Sep. 2003.
[13] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *MOBICOM*, 2001.
[14] Mesh Networks Inc. http://www.meshnetworks.com.
[15] T. Nandagopal, T. Kim, X. Gao, and V. Bharghavan. Achieving MAC Layer Fairness in wireless packet networks. In *MOBICOM*, 2000.
[16] A. Rainwala and T. Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *INFOCOM*, 2005.
[17] MIT roofnet. http://www.pdos.lcs.mit.edu/roofnet/.
[18] K. Xu, M. Gerla, and S. Bae. How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks? In *GLOBECOM*, 2002.
[19] X. Yang and N. H. Vaidya. Priority scheduling in wireless ad hoc networks. In *MOBIHOC*, June 2002.
[20] M. Yarvis, K. Papagiannaki, and W. S. Conner. Characterization of 802.11 wireless networks in the home. In *WiNMeE*, 2005.

Exploiting Partially Overlapping Channels in Wireless Networks: Turning a Peril into an Advantage

Arunesh Mishra [α], Eric Rozner [β], Suman Banerjee [β], William Arbaugh [α]
[α] *University of Maryland, College Park.* [β] *University of Wisconsin, Madison.*

Abstract

Interference has always been considered as an unavoidable peril in wireless networks. A single data transmission is useful to some nodes and becomes interference to others. Based on channel of origin, interference can be categorized into co-channel (from transmissions on the same channel as the receiver) and adjacent-channel (transmissions on adjacent and overlapping channels).

In this paper, we define specific mechanisms that can transform partially overlapped channels into an advantage, instead of a peril. We construct simple analytical and empirical models of such interference occurring in IEEE 802.11 networks, and illustrate two scenarios where such interference can be exploited. First, we apply partially overlapping channels to improve spatial channel re-use in Wireless LANs (WLANs). Second, we leverage such channels to enable nodes with a single radio interface to communicate more efficiently with their peers in 802.11 ad-hoc mode potentially using multi-hop paths. We evaluate both capabilities through testbed measurements.

1 Introduction

The IEEE 802.11 b/g standards operate in the unlicensed ISM 2.4 Ghz spectrum which has 11 out of 14 channels available for use in the US. The channel number $(1 \ldots 11)$ represents the center frequency on which the radios operate (e.g. 2.412 Ghz for channel 1). The center frequencies are separated by 5 MHz, while the channels have a spread of about 30 MHz around the center frequency. As a result, a signal on any 802.11b channel overlaps with several adjacent channels, with the *extent* of overlap decreasing with increasing separation between the center frequencies. Attributing to this overlap, a transmission on one channel becomes interference to stations on an overlapping channel, also known as adjacent channel interference.

The term adjacent channel interference can be contrasted to co-channel interference which occurs from transmissions on the same channel. In 802.11 and other wireless networks, adjacent channel interference is considered a peril. In order to avoid this peril, two simultaneously communicating nodes that are in close proximity are assigned to different non-overlapping channels, i.e., channels 1, 6, and 11 in 802.11b are non-overlapping. For example, in a inbuilding WLAN with multiple APs, interfering APs are assigned to different non-overlapping channels. Traditionally, in the literature on channel assignment algorithms, the notion of availability of a set of N channels has inherently implied that they are non-overlapping. This is why the number of available channels in 802.11b networks is considered to be three. However, many channel pairs in the ISM band overlap partially. Given that wireless spectrum is a scarce resource, in this paper we discuss the basic concepts behind building techniques to carefully exploit the partially-overlapped nature of channels for efficient spectrum management.

Consider a simple experiment where a transmitting station is placed on channel 6, and a receiving station is moved from channel 1 through 11. The two stations are placed in close proximity to each other to preclude effects of signal attenuation due to distance. Table 1 shows the signal-to-noise ratio (SNR) of the received signal on channels $1 \ldots 11$. The SNR is normalized to a scale of $0 \ldots 1$ with 1 denoting the maximum signal received and 0 the minimum (indicating background noise level). We can see that the interference between channels 1 and 6, and channels 6 and 11 is minimum – hence they are considered non-overlapping. However, there exist other channel pairs (eg:- $\langle 2, 6\rangle$ and $\langle 6, 10\rangle$) where the interference is fairly low.

We define a term *Interference-factor* or simply *I-factor* denoted by $I(i, j)$ as the *extent* of overlap between channels i and j. If P_i denotes the power received at a given location of a particular signal, and P_j denote the power received of the same signal at the same location on channel j, then $I(i, j)$ is defined as $\frac{P_i}{P_j}$. $I(i, j)$ gives the fraction of a signal's power on channel j that will be received on channel i. Table 1 thus shows $I(i, 6)$ normalized to a scale of $0 \ldots 1$. I-factor can be calculated analytically as well as empirically and does not depend on the radio propagation properties of the environment (i.e. open space or indoors). It depends on the extent of frequency overlap between the signals on channels i and j.

We believe that ours is the first systematic effort on taking advantage of partial overlap among channels in two different settings — WLANs and mesh networks. Each scenario illustrates a novel conceptual notion of ex-

Channel	1	2	3	4	5	6	7	8	9	10	11
Normalized SNR (I-factor)	0	0.22	0.60	0.72	0.77	1.0	0.96	0.77	0.66	0.39	0

Table 1: Table shows the signal-to-noise ratio (SNR) normalized to a scale of $0 \ldots 1$ of the transmission made on channel 6 as received on channels $1 \ldots 11$. We call this quantity *I-factor*.

ploiting the overlap among channels. To achieve some desired benefits we will leverage the following two observations. 1) As shown in Table 1 partial overlap among two channels reduces the signal strength of a transmission on one channel as received on the other (when compared to using the *same* channel). This improves the *spatial re-use* of channels which refers to how frequently a single channel can be re-used without interference. 2) Partially overlapped channels will allow communication between two nodes operating on non-overlapped channels, by forwarding traffic among them. We describe the two settings next.

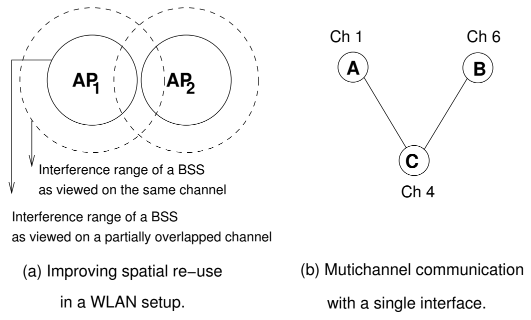

AP1
AP2
Interference range of a BSS as viewed on the same channel
Interference range of a BSS as viewed on a partially overlapped channel
(a) Improving spatial re-use in a WLAN setup.
Ch 1
Ch 6
A
B
C
Ch 4
(b) Mutichannel communication with a single interface.

Figure 1: Two advantages of using partial overlapping channels in wireless networks.

WLAN scenario: In a WLAN, an AP and its associated clients which form a basic service set (BSS), communicate on a single channel. Two APs can operate on the same channel as long as they do not interfere with each other. Better spatial re-use of channels brings about improvement in the network's throughput capacity as a whole [9]. Specifically in a WLAN, this allows the placement of more APs without interference and thus providing better service to an increased number of clients.

Neighboring APs in a WLAN are typically spaced sufficiently far apart so as to maximize the region of coverage while allowing some overlap for handoffs to occur smoothly. Figure 1(a) shows two APs and the interference range of their BSS which is the distance upto which transmission from the AP or an associated client would interfere (circular ranges for illustration only). Here these APs will have to be assigned to non-overlapping channels, or the APs will have to be spaced further apart to allow them to operate on the same channel. Instead of assigning them to non-overlapping channels, one can assign them partially overlapping channels such that the signal attenuation due to the partial overlap makes the interference negligible (i.e. the signal strength degrades to a tolerable level). Conceptually, this can be thought of as reduction in the interference range of an AP's BSS as viewed by a receiver on an adjacent overlapping channel. The reduction in the interference range increases with increased separation among the channels, with complete reduction perceived over non-overlapping channels.

It may appear that similar reduction in BSS size can be achieved by reducing transmission power of the APs. Unfortunately such a change does not lead to the same level of spectral use for two reasons. 1) Although by reducing transmission powers in 802.11b networks APs can be brought as close to each other as in the adjacent channel case, the number of channels that can be utilized is still three (and not four). 2) Reduction in transmission power reduces the signal to noise ratio in the AP-client interaction and adversely affects performance for clients. Hence, by exploiting adjacent channel interference we can get much better performance than by just reducing transmission power at APs. We evaluate this effect over TCP/UDP throughputs and MAC level collisions as measured in testbed environments in Section 3.

Mesh networks scenario: Consider a very simple mesh network setting with three nodes each having just one radio interface (Figure 1(b)). In general, if node A wants to communicate with node B, both these nodes need to use the same channel. However, let us assume that due to assignment of channels to other neighboring nodes, A and B need to be assigned to channels 1 and 6 respectively. However, if node C can be assigned to channel 4, then there are partial overlap between channels assigned to A and C as well as between B and C. Hence C can route traffic from A to B, using the partially overlapping properties of channels. Such opportunistic mechanisms can have non-trivial impact on the routing system used by the mesh network. In general, mesh network nodes where the number of interfaces available is less than the number of non-overlapping channels provided by the physical layer (such as 12 in 802.11a) can utilize such optimizations to communicate with nodes on multiple channels providing increased flexibility to the routing infrastructure used by the network. We evaluate this using experiments in Section 4. Next section presents the analytical basis behind spatial re-use.

2 Analytical Basis behind Spatial Re-use

We develop analytical reasoning behind using partially overlapping channels to improve spatial re-use of spectrum. We borrow some of the terminology used in [10] for the discussion below.

For the simplicity of discussion, we will assume an open-space environment, in which the path loss of a signal is usually modeled by a two-ray ground propagation model [7]. Let d is the distance between the receiver and the transmitter. According to this model, in the *open space* environment, the received power P_r of a signal from a sender at distance d is given by

$$P_r = P_t G_t G_r \frac{h_t^2 h_r^2}{d^k} \quad (1)$$

In Equation 1, G_t and G_r are the antenna gains of the transmitter and the receiver respectively. h_t and h_r are the height of both antennas. P_t is the transmission power at the sender. The path loss, parameter k, typically is a value between 2 and 4.

A signal arriving at a receiver can be demodulated correctly if the *signal-to-noise ratio(SNR)* is above a certain threshold (say T_{SNR}). Say, a transmitter T and receiver R are separated by distance d, while another sender S (potentially causing interference) is at a distance r from the receiver R and on the same channel. Let P_d denote the receiving power of the signal from the transmitter T, and P_r the power of the signal from S at R. Thus, the SNR is given as $SNR = \frac{P_d}{P_r + N_a}$, where N_a is the ambient or thermal noise in the environment around the receiver R. For simplicity of analysis, we neglect N_a compared to P_r, and assume homogeneous radios. We have,

$$SNR = P_d/P_r = \left(\frac{r}{d}\right)^k \geq T_{SNR} \quad (2)$$

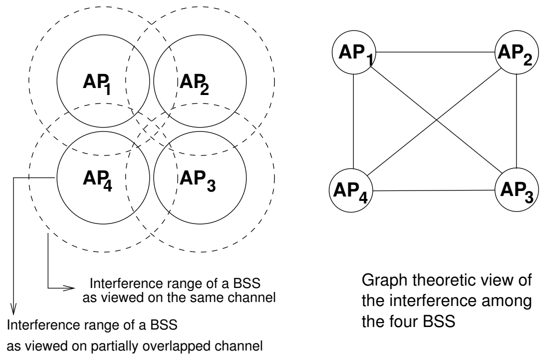
$$r \geq \sqrt[k]{T_{SNR}}.d \quad (3)$$

Thus, to successfully receive a signal, the interfering nodes must be at least $\sqrt[k]{T_{SNR}}.d$ distance away from the receiver.

Now, if T and R were on channel i and the sender was on channel j, the minimum separation between S and R would become $\sqrt[k]{T_{SNR} I(i,j)}.d$, where $I(i,j)$ denotes the I-factor (see Table 1) as defined in Section 1. Note that since $I(i,j) \leq 1$, this new distance is less than $\sqrt[k]{T_{SNR}}d$, thus allowing the stations to get closer without incurring interference. This improves the spatial reuse of the wireless spectrum. We show how this concept can improve channel assignment in WLANs next.

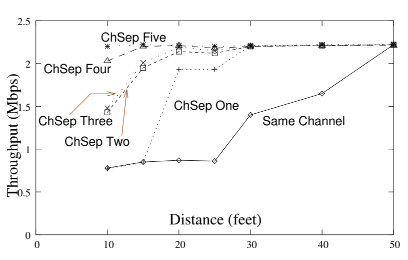
3 Channel Assignment in WLANs

Channel assignment to APs in a wireless LAN is typically performed statically by the network adiminstrators after performing an RF site survey, or autonomously by the APs performing a "least congested channel search (LCCS)", i.e., identifying a channel which has low interference. Both approaches have been shown to have a number of inefficiencies, and other dynamic and distributed approaches have been proposed [4, 5]. Here we show using a simple example how partially overlapping channels can yield improved network throughput and can be complimentary to existing channel assignment methods.

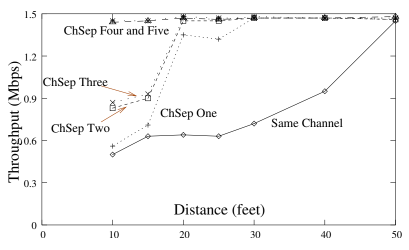

AP1
AP2
AP4
AP3
Interference range of a BSS as viewed on the same channel
Interference range of a BSS as viewed on partially overlapped channel
Graph theoretic view of the interference among the four BSS
Figure 2: Improving spatial re-use of spectrum with partially overlapping channels in a WLAN environment.

Consider the WLAN setup shown in Figure 2. The dashed circle shows the size of an AP's cell or the basic service set (BSS). This region is the distance upto which a transmission from either the AP or a client associated to the AP would interfere with stations on the same channel as the AP (circular ranges for ease of illustration). There are four APs whose BSS have reasonable overlap. Network administrators would organize APs in this manner so as to maximize total coverage while allowing reasonable overlap for handoffs to occur. If there are large number of clients in the common region of overlap, the four APs would need to be assigned to different channels. Such constraints are typically captured in a graph theoretic manner as shown in the figure, and channel assignment becomes the well-known graph coloring problem. The example of Figure 2 becomes a 4-clique, which needs four channels to eliminate interference from neighboring APs. This is impossible to achieve in the 802.11b system, which has only 3 non-overlapping channels.

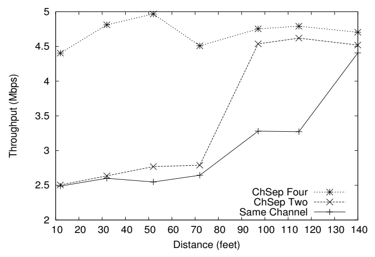
Instead of using 3 non-overlapping channels, the APs could use partially-overlapped channels, e.g., 1, 4, 7 and 11. The partial overlap among these channels would be sufficient to degrade the signal below an acceptable level so as to not cause interference. This is conceptually viewed as reducing the interference range of a BSS by a certain factor as observed by a receiver on a neighboring channel – the *extent* of overlap among two channels which we defined as the *I-factor*. Note that the range of a BSS itself (as viewed in the same channel) is not altered. Hence clients associated with this AP see no difference in performance, while clients of neighboring APs on partially overlapped channels see reduced interference. Thus, by using these four partially overlapping



(a) UDP throughput



(b) TCP throughput



(c) TCP throughput at 11Mbps in a second office-building environment.

Figure 3: TCP/UDP throughputs versus spatial separation (*ChSep*).

channels, the respective BSS get rid of any interference from neighboring APs

Below we discuss a systematic method for taking advantage of the partially overlapping channels in an algorithmic manner. We note that at this point this is work-in-progress as we work towards a full-fledged solution:

Let k denote the total number of non-overlapping channels available in the underlying wireless PHY layer. Given a region of interest covered with a set of access points, define an *overlap* graph $G = (V, E)$ as follows: $V = \{ap_1, ap_2, \ldots, ap_n\}$ be the set of n APs that form the network. Place an edge between APs ap_i and ap_j ($ap_i \neq ap_j$) if the users associated to the respective APs or the APs themselves interfere (indicating spatial overlap in region of coverage).

Assume we are given a channel assignment that uses the 3 non-overlapping channels in 802.11b. This assignment could have been performed using LCCS or other techniques [5]. Reflect the channel assignment onto the above constructed overlap graph. We say that an edge (ap_i, ap_j) is a *conflict* edge if there is interference between the respective APs – i.e. they are assigned the same channel. Consider such a conflict edge (ap_i, ap_j). Note that both ap_i and ap_j, have neighbors utilizing the other two channels (else this conflict can be removed). Now, we apply the partially overlapping channels to this subgraph formed by ap_i, ap_j and all neighbors of ap_i and ap_j, we observe if the partial overlap can reduce the interference (as discussed in the example above). This intuition can be used to algorithmically use partially overlapping channels.

Next, we discuss detailed experimental results which show improved spatial reuse of spectrum using partially overlapping channels.

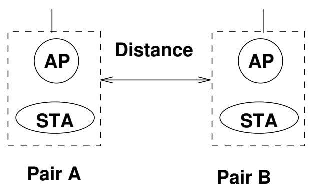


Figure 4: The experimental setup used.

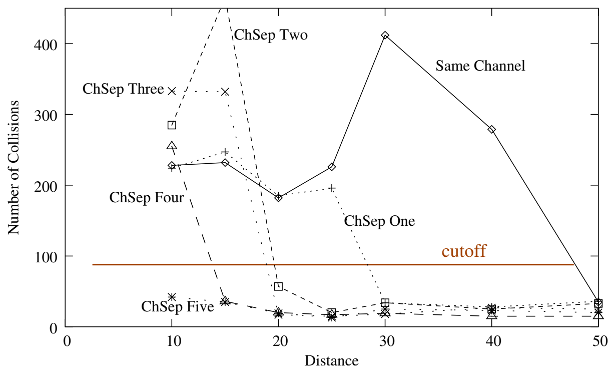


Figure 5: Number of collisions versus spatial separation.

3.1 Experiments to demonstrate spatial reuse

To demonstrate how partially overlapping channels can be used to improve reuse of the RF spectrum, we consider the following setup. Two APs and two stations (STA), with one STA associated to each AP, take part in the experiments. One such AP-STA pair, called *Pair-A*, is kept fixed at a particular location within an office building, while the other *Pair-B* is moved to various different locations of measurement. The distance between an AP and its associated STA is kept constant throughout the experiment process.

The following parameters are varied to study the effect of using partially overlapping channels on TCP/UDP throughput and MAC level collisions: The distance between the AP-STA pairs is varied to study the interfer-

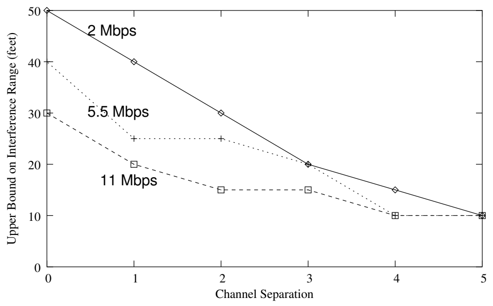


Figure 6: Interference ranges vs channel separation for datarates of 2, 5.5 and 11 Mbps.

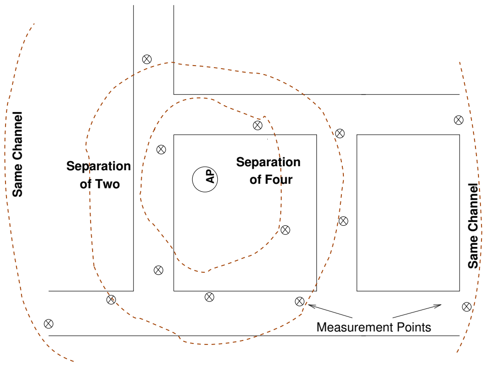


Figure 7: Measurement setup used along with interference ranges at channel separation of 0 (same channel), 2 and 4.

ence range of the BSS formed by the pair. The datarate used for communication was chosen from the following permissible instantaneous data rates: 2, 5.5 and 11 Mbps (note, throughputs will be lower). The channel separation, which refers to the difference in the channel numbers used by the two pairs, is varied between 0 and 5. For each selection of the parameters, we monitored TCP and UDP throughputs for flows lasting 10 seconds. The MAC level collisions were also monitored. All APs and STAs used wireless NICs from the same vendor with a constant transmit power of 30mW.

Figure 3(a) plots the UDP throughput achieved by the AP-STA Pair-B against distance from Pair-A. This experiment used a datarate of 2 Mbps which best shows how a step by step decrease in overlap increases the AP-STA throughput (other datarates omitted due to space restrictions). Figure shows that using the same channel, a distance of around 50 feet would be necessary to eliminate the interference and attain the maximum possible throughput. This throughput is attained for much smaller distances as the channel separation (indicated by the legend *ChSep*) is increased from 0 (same channel) to 5 (non-overlapping channels). Figure 3(b) shows the same effect on TCP throughput. Figure 3(c) shows the TCP throughput in a second environment, which is an office building with a long corridor on which the experiments were performed. The datarate here was set at 11 Mbps.

Figure 5 shows the number of collisions that occurred at the AP-STA Pair-B. The number of collisions have a significant amount of variation from one execution of the experiment to another because of the randomness present in the MAC protocol. However, clearly noticeable is the fact that a drastic reduction in the number of collisions (as shown by the *cutoff* at around 100 in Figure 5) indicates significant reduction in interference. Thus, the points at which each curve takes a plunge below the *cutoff* line (shown in the figure) indicates the distance at which interference does not occur between the two AP-STA pairs.

Figure 6 shows the effect of the 802.11 MAC datarate on the interference range of a BSS. Each point on a curve plots the minimum observed distance (modulo discrete observation points) on the y-axis at which with the given channel separation (x-axis) the two AP-STA pairs do not interfere with each other and attain the maximum possible TCP/UDP throughput – which is the interference range of the BSS formed by an AP-STA pair. One observes that for all three datarates, the interference range decreases consistently. Also as expected, for a given amount of channel overlap, a higher datarate has a smaller interference range.

Figure 7 shows graphically the interference ranges for channel separation of zero (same channel), 2 and 4 at a datarate of 2 Mbps. This visually demonstrates how spatial reuse can be significantly improved by carefully employing partially overlapping channels.

4 Applications to Mesh Networks

Proper channel assignment and routing are important to utilize the full capacity of a mesh network. Channel assignment affects the topology that is available to the routing infrastructure and these problems are addressed in tandem with each other [6, 2, 8, 3]. Also, there are techniques which utilizes a single wireless card to connect to multiple networks by constantly switching channels [1]. Partially overlapping channels that allow communication with nodes operating on different non-overlapping channels can yield significant advantages in such settings. They can be employed to connect to multiple networks, or to add flexibility to the routing infrastructure by creating additional edges in the mesh network topology. Below we demonstrate simple experiments which show how two nodes operating on partially overlapping channels can communicate with each other.

We conduct the following simple experiment: Two nodes are placed on channels with decreasing overlap, and the UDP throughput is measured. One node was kept fixed on channel 6, while the other was progressively moved from channel 1 through 11. The nodes were configured at a datarate of 2Mbps. Figure 8 shows the plots at distances of 15 and 30 feet between the two nodes. The plots demonstrate how a partially overlapping channels can be used to communicate at the cost of reduction in the throughput. This allows a node with a single interface to communicate with nodes on two non-overlapping channels by operating on a channel that partially overlaps with both.

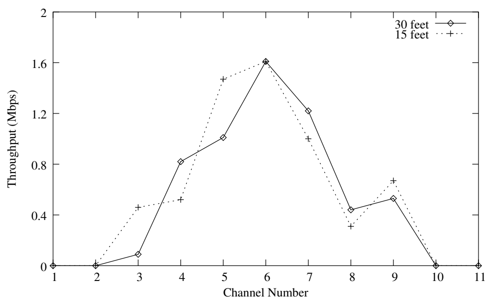


Figure 8: Throughput versus channel separation.

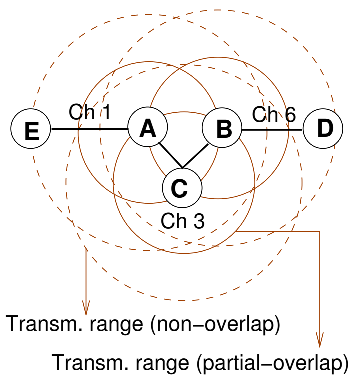


Figure 9: Example multihop scenario.

Throughput improvements using partially overlapping channels: Apart from utilizing partially overlapping channels for topology flexibility in mesh settings, they can be applied to improve the throughput capacity of multi-hop networks. Consider the multihop setup shown in Figure 9. There are four nodes; each with a single radio interface. Because of this limitation, technically the network can utilize only one channel or the network would get partitioned. Say, nodes A, B and C are within communication range of each other. Nodes D and E are each within communication range of nodes A,B and C. The transmission ranges of nodes A,B and C are shown using dashed circles. With all nodes operating on a single channel to prevent network partition, each node shares the channel with two other nodes, thus achieving roughly $1/3$rd of the maximum available bandwidth on one channel.

Now, using partially overlapping channels, an assignment can be performed as follows. Nodes A and E are assigned channel 1; nodes B and D are on channel 6. These links do not interfere with each other. Node C is be placed on channel 4, which allows communication with both A and B. Channel 4 is chosen such that the reduction in the communication range as determined by the extent of overlap between $\langle 4, 6\rangle$ and $\langle 4, 1\rangle$ allows communication with both A and B. The network thus utilizes the maximum bandwidth available on two non-overlapping channels and hence this can result in significant throughput improvements.

5 Summary

Through testbed measurements we have evaluated how partial overlap in channels can be exploited for improved spatial re-use (WLANs) and the ability to do multichannel communication (Mesh). The focus of this paper has really been to add a new mechanism to our toolkit for spectrum management — partially overlapped channels. We believe this paper is merely the first step towards design of new and efficient algorithms that are aware of the potential demonstrated. In particular it would be possible to re-visit each channel assignment technique designed in the past and examine how they can be extended by employing this new mechanism.

References

[1] Chandra, R., Bahl, P., and Bahl, P. Multinet: Connecting to multiple ieee 802.11 networks using a single wireless card. In *Proceedings of IEEE Infocom* (2004).

[2] Draves, R., Padhye, J., and Zill, B. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of ACM Mobicom* (2004).

[3] Kyasanur, P., and Vaidya, N. Routing and interface assignment in multi-channel multi-interface wireless networks. In *Proceedings of IEEE WCNC* (2004).

[4] Lee, Y., Kim, K., and Choi, Y. Optimization of ap placement and channel assignment in wireless lans. In *Proceedings of 27th Annual IEEE Conference on Local Computer Networks (LCN)* (2002).

[5] Mishra, A., Banerjee, S., and Arbaugh, W. Weighted coloring based channel assignment for wlans. *ACM SIGMOBILE Mobile Computer Communications Review* (2005).

[6] Raniwala, A., and cker Chiueh, T. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *Proceedings of IEEE Infocom* (2005).

[7] Rappaport, T. *Wireless Communications: Principle and Practice.* Prentice Hall, 1996.

[8] So, J., and Vaidya, N. Routing and channel assignment in multi-channel multi-hop wireless networks with single network interface. *Technical Report, University of Illinois at Urbana Champaign* (2005).

[9] Stine, J. A., de Veciana, G., Grace, K. H., and Durst, R. C. Orchestrating spaital reuse in wireless ad hoc networks using synchronous collision resolution. *World Scientific Journal of Interconnection Networks* (2003).

[10] Xu, K., Gerla, M., and Bae, S. How effective is ieee 802.11 rts/cts handshake in ad hoc networks? In *Proceedings of IEEE GLOBECOM* (November 2002).

Network Anomography

Yin Zhang[†], Zihui Ge[‡], Albert Greenberg[‡], and Matthew Roughan[§]
[†]*Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA*
[‡]*AT&T Labs – Research, Florham Park, NJ 07932, USA*
[§]*School of Mathematical Science, University of Adelaide, SA 5005, Australia*

Abstract

Anomaly detection is a first and important step needed to respond to unexpected problems and to assure high performance and security in IP networks. We introduce a framework and a powerful class of algorithms for *network anomography,* the problem of inferring network-level anomalies from widely available data aggregates. The framework contains novel algorithms, as well as a recently published approach based on Principal Component Analysis (PCA). Moreover, owing to its clear separation of inference and anomaly detection, the framework opens the door to the creation of whole families of new algorithms. We introduce several such algorithms here, based on ARIMA modeling, the Fourier transform, Wavelets, and Principal Component Analysis. We introduce a new *dynamic anomography* algorithm, which effectively tracks routing and traffic change, so as to alert with high fidelity on intrinsic changes in network-level traffic, yet not on internal routing changes. An additional benefit of dynamic anomography is that it is robust to missing data, an important operational reality. To the best of our knowledge, this is the first anomography algorithm that can handle routing changes and missing data. To evaluate these algorithms, we used several months of traffic data collected from the Abilene network and from a large Tier-1 ISP network. To compare performance, we use the methodology put forward earlier for the Abilene data set. The findings are encouraging. Among the new algorithms introduced here, we see: high accuracy in detection (few false negatives and few false positives), and high robustness (little performance degradation in the presence of measurement noise, missing data and routing changes).

1 Introduction

The first step in fixing a problem is knowing it exists. This is no less true in networking than anywhere else – we need to know about a problem before we can repair it. Networking vendors typically build alarms into network equipment to facilitate fast, accurate detection and diagnosis of problems. However, in practice, there are many problems for which explicit alarms are either absent (for new or uncommon problems), or intrinsically hard to produce. In these cases we must infer the problem from other data sources. For instance, many types of network problems cause abnormal patterns to appear in the network traffic. Such traffic *anomalies* may be caused by problems ranging from security threats such as Distributed Denial of Service (DDoS) attacks and network worms, to unusual traffic events such as flash crowds, to vendor implementation bugs, to network misconfigurations. We refer to the problem of inferring anomalies from indirect measurement as *network anomography* (combining "anomalous" with "tomography," a general approach to such inference problems).

Network tomography [31] bears some resemblance, in that both involve the solution of a linear inverse problem. Examples include inference of individual link performance characteristics from path performance characteristics, and inference of traffic matrices from individual link load measurements. For example, the traffic matrix estimation problem arises because the obvious source of data for direct measurement (flow-level data) can be hard to obtain network-wide [5, 14, 22, 23, 28, 31, 34, 35]. On the other hand, Simple Network Management Protocol (SNMP) data on individual link loads is available almost ubiquitously. Fortunately, the link loads and traffic matrices are simply related by a linear equation

$$\mathbf{b} = A\mathbf{x} \tag{1}$$

The vector $\mathbf{b}$ contains the link measurements, and A is the routing matrix (defined formally below). We wish to infer $\mathbf{x}$, which contains the unknown traffic matrix elements written as a vector. Tomographic inference techniques seek to invert this relationship to find $\mathbf{x}$.

The anomography problem is different and somewhat more complex. First, note that anomaly detection is performed on a series of measurements over a period of time, rather than from a single snapshot. In addition to changes in the traffic, the solution must build in the ability to deal with changes in routing. Second, note that the anomalies that we wish to infer may have dramatically different properties from a traffic matrix, and so different methods than those used for network tomography may be called for. Indeed, we find that simple extensions to network tomography methods perform fair poorly here. Techniques that transform the measurements prior to attempting to solve the inverse problem are preferable.

As a simple example, imagine trying to detect an anomalous traffic pattern caused by a flash crowd or DDoS attack on a web site. This type of event will cause increases in traffic flows headed towards a particular set of destinations. It may be hard to rapidly identify which of the tens of thousands of ingress links on a large network might be primarily responsible, as large surges at a network egress link may arise from small surges on several ingress links. We must infer the change in the pattern of traffic to the particular site from the complete set of link data, considered together, rather than as individual time series. This illustrates an important feature of anomography – it extends anomaly detection to network-level problems (automatically building in correlation across the network) where link-level anomaly detection might be inadequate or unreliable.

Many approaches to anomography are possible. In pioneering work, Lakhina *et al.* introduced a novel approach based on Principal Component Analysis (PCA) [19]. Our paper makes three major contributions to understanding and solving anomography problems:

1. We present a simple and powerful framework that encompasses a wide class of methods for network anomography. We will see that the method of [19] is a member of this class. The framework clearly decouples the inference and anomaly detection steps, and so immediately opens the door to the development of new algorithms where one makes different choices for each step. Accordingly, we introduce several such new algorithms here, based on ARIMA modeling, the Fourier transform, Wavelets, and Principal Component Analysis. Moreover, the framework is not restricted to the analysis of link traffic data, and in particular also applies to the dual problem of inferring performance anomalies from end-to-end performance measurements.
2. We introduce a new algorithm for *dynamic anomography*, which identifies network level traffic anomalies and works in the presence of routing changes. That is, dynamic anomography tracks routing and traffic change – signaling traffic anomalies, but not internal network routing changes (which may dramatically change internal traffic patterns but may leave the traffic matrix, describing how traffic enters and exits the network, stable). In IP networks, routing changes occur as part of the normal "self-healing" behavior of the network, and so isolating these from traffic anomalies is advantageous. An additional benefit of dynamic anomography is that it is robust to missing link load measurements, an important operational reality (see Section 4 for why missing data may result in changes in the routing matrix). To the best of our knowledge, this is the first anomography algorithm that can handle routing changes and missing data.
3. Using data sets collected from a large Tier-1 ISP and from Internet2's Abilene network, we report on the results of an extensive and thorough evaluation of a set of anomography methods. To understand the fidelity of the methods and to compare different methods, we apply the methodology introduced in [19]. Under this methodology, we find that in general the new *temporal anomography* methods introduced here exhibit consistently high fidelity. In particular, we find that the most successful method (of those examined) is a variation of dynamic anomography, combining Box-Jenkins modeling (ARIMA) with ℓ^1 norm minimization. Further evaluation suggests that this algorithm can cope well with measurement noise, and degrade gracefully in the presence of missing or corrupted data.

The paper is organized as follows. Section 2 summarizes background and related work. In Section 3 we describe our framework, and the anomography algorithms examined in this paper, in the context of fixed routing. In Section 4 we extend the Box-Jenkins anomography to the case where routing may change over time. In Section 5 we describe our evaluation methodology, and Section 6 presents the results. Section 7 provides final remarks.

2 Background

2.1 Network Tomography

Network tomography describes several problems: inferring link performance of a network from end-to-end measurements, or inferring Origin-Destination (OD) traffic demands from link load measurements. These problems can be written as linear inverse problems where one seeks to find unknowns $\mathbf{x}$ from measurements $\mathbf{b}$ given a linear relationship (1), where A is the routing matrix. For a network with n links, and m OD flows, we define the routing matrix to be the $n \times m$ matrix $A = [a_{ij}]$ where a_{ij} indicates the fraction of traffic from flow j to appear on link i.

SNMP provides link measurements of traffic volumes (bytes and packets), typically at 5 minute intervals (this data is described in more detail in, for example [34]). We shall assume data of this type is the input to our algorithms, and we wish to infer anomalous traffic matrix elements, but note that anomography is not limited to this problem, and could equally be applied to inferring anomalous link performance from end-to-end measurements. An additional source of data used here comes from the routing protocols used to build the forwarding tables within each router. We use routing data (e.g., gathered from a route monitor as in [27]) along with a route simulator (as in [10]) to predict the results of these distributed computations, and determine the network routing.

The problem of inferring the OD traffic-matrix has been much studied recently (*e.g.*, [5, 14, 22, 23, 28, 31, 34, 35]). The problem's key characteristic is that it is massively underconstrained: there will be approximately N^2 OD flows to estimate and only $O(N)$ link measurements. Hence tomography methods seek to introduce additional information, often in the form of some kind of traffic model (for instance a Poisson model in [31, 28], a Gaussian model in [5], or a gravity model in [34, 35]). Anomography problems are also highly underconstrained, but the models used to describe traffic are inappropriate for anomalies — by definition these events are generated by completely different processes from normal network traffic. Moreover, in anomography we combine detection with inference, whereas in standard network tomography we seek only to infer a set of traffic matrix elements. Hence there are important differences between this paper and network tomography.

It is also important to note that routing matrices change over time. In much previous work, routing matrices are taken to be constant (an exception being [23], where the traffic is assumed to be somewhat constant, while the routing varies), but it is important (see [29]) to allow for the fact that routing is not constant, and neither is the traffic. In order to allow for variable routing, we index not just the traffic measurements over time, but also the routing matrix. Given these, we may write the relationship between the link traffic, and OD traffic matrix as

$$\mathbf{b}_j = A_j \mathbf{x}_j, \tag{2}$$

where A_j is an $n \times m$ routing matrix, $\mathbf{x}_j$ is a length-n vector of unknown OD flow traffic volumes, and $\mathbf{b}_j$ is a length-m vector of link loads[1], at time interval j.

2.2 Related Work

Lakhina *et al.* carried out the pioneering work in the area of inference of anomalies at network level, [19, 18, 20], and adapted Principal Components Analysis (PCA) to this setting. Donoho [8, 9] introduced a powerful mathematical treatment for tomography-like problems, wherein one seeks solutions that maximize sparsity (intuitively, solutions with fewest explanations). These papers inspired our development of the new methods introduced here, and our development of a framework in which a very wide class of methods all fit.

Anomaly detection is a burgeoning field. A great deal of research in network anomaly detection relies on some type of inference step, taking a set of alarms [13, 15, 16, 25, 30] as input. While anomography includes methods of this type, our results indicate that it is better to delay alarm generation until after the inference step. In that way, a single constructive alarm may be generated, rather than a storm of redundant alarms. Moreover, in delaying the alarm generation until after the inference step, we can in some cases greatly improve the sensitivity of detection, as was demonstrated in [19].

We approach the network anomaly detection problem from the point of detecting unknown anomalous behavior, rather than looking for particular signatures in the data, the focus of much work in the security community. A large component of the work on machine learning, signal processing and time-series analysis is devoted to detecting outliers or anomalies in time-series. This literature has been applied to networks in a number of cases; for examples see [1, 4, 15, 17, 30, 32]. These methods range in sophistication from [4], which suggests the use of the standard Holt-Winters forecasting technique for network anomaly detection, to [1], which uses a sophisticated wavelet based method with great potential. These methods focus on single time series rather than the multi-dimensional time series that arise in anomography.

Most earlier work ignores noise or provides weak tests of robustness to noise (which can destroy utility). A strength of the work presented here is that we provide tests of effectiveness of the methods in the presence of noise, always a factor in practice.

3 Network Anomography

In this section, *we shall assume that the routing matrices A_j are time-invariant and are denoted by A.* (We will extend our work to time-varying A_j in Section 4.) Under this assumption, we can combine all t linear systems (2) into a single equation using matrix notation:

$$B = AX, \tag{3}$$

where $B = [\mathbf{b}_1 \, \mathbf{b}_2 \cdots \mathbf{b}_t]$ is the matrix formed by having $\mathbf{b}_j$ as its column vectors, and similarly $X = [\mathbf{x}_1 \, \mathbf{x}_2 \cdots \mathbf{x}_t]$.

3.1 A General Anomography Framework

We identify two basic solution strategies to network anomography: (i) *early inverse*, and (ii) *late inverse*. Early-inverse approaches may appear more intuitive. The early-inverse approach tackles the problem in two steps. The first is the *network tomography* step, where OD flow data at each interval j are inferred from the link load measurements by solving the ill-posed linear inverse problem (2). Given the estimated OD flow data $\mathbf{x}_j$ at different time points j, in the second step, *anomaly detection* can then be applied to the $\mathbf{x}_j$. For this step, there are many widely used spatial and temporal analysis techniques, which we will describe later in this section.

Early-inverse methods, although conceptually simple, have an obvious drawback — errors in the first step, which are unavoidable due to the ill-posed nature of the inference problem, can contaminate the second step, sabotaging overall performance. Another disadvantage is that early-inverse methods apply a potentially computationally expensive anomaly detection step to high-dimensional data: on a network of N nodes, one must perform this step on all N^2 OD pairs. As we will see, late-inverse performs anomaly detection on only $O(N)$ dimensional data. We focus on late-inverse methods in this paper for these reasons, though we shall provide some comparisons between early- and late-inverse methods.

The idea of the late-inverse method is to defer "lossy" inference to the last step. Specifically, late inverse approaches extract the anomalous traffic from the link load observation, then form and solve a new set of inference problems:

$$\tilde{B} = A\tilde{X}, \tag{4}$$

where $\tilde{B} = [\tilde{\mathbf{b}}_1 \, \tilde{\mathbf{b}}_2 \cdots \tilde{\mathbf{b}}_t]$ is the matrix of anomalous traffic in the observables, and $\tilde{X} = [\tilde{\mathbf{x}}_1 \, \tilde{\mathbf{x}}_2 \cdots \tilde{\mathbf{x}}_t]$ is the matrix of OD flow anomalies to be diagnosed, over t time intervals.

While the new inference problems (4) share the same linear-inverse structure as in network tomography (3), the characteristics of the unknowns are very different, and so is the solution strategy, which we will explore in Section 3.4.

We now introduce a simple framework for late-inverse anomography methods. In this framework, $\tilde{B}$ is formed by multiplying B with a transformation matrix T. Depending on whether we use a left or right multiplying transformation matrix, we can further divide the framework into the following two classes:

- *spatial anomography*, where a left multiplying transformation matrix T is used to form $\tilde{B}$, *i.e.*, $\tilde{B} = TB$;
- *temporal anomography*, where a right multiplying transformation matrix T is used to form $\tilde{B}$, *i.e.*, $\tilde{B} = BT$.

Our framework encompasses a number of analysis techniques for extracting anomalous traffic $\tilde{B}$ from link load observations B, as we next illustrate.

3.2 Spatial Anomography

Data elements in high dimensional data sets, such as the link load observations, usually have dependencies. The intrinsic dependency structure among the data elements can thus be exploited for filtering anomalous behavior by discovering data points that violate the normal dependency structure. In our context, the process of detecting such data points can be performed by left-multiplication by a transformation matrix T such that $\tilde{B} = TB$. An example of such an approach is a recent study by Lakhina *et al.* [19], where Principal Component Analysis (PCA) is used in finding dominant patterns. We describe this method, and in particular its instantiation as a left-multiplication operation in the following section.

3.2.1 Spatial PCA

In [19], Lakhina *et al.* proposed a subspace analysis of link traffic for anomaly detection, which can be summarized as follows.

1. Identify a coordinate transformation of B such that the link traffic data under the new coordinate systems have the greatest degree of variance along the first axis, the second greatest degree of variance along the second axis, and so forth. These axes are called the principal axes or principal components.
   Recall that $B = [\mathbf{b}_1\, \mathbf{b}_2 \cdots \mathbf{b}_t]$ is the collection of link traffic data at m links over t time intervals, where each row i ($1 \leq i \leq m$) denotes the time series of the i-th link and each column j ($1 \leq j \leq t$) represents an instance of all the link loads at time interval j. The principal components, $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m$ can be computed iteratively as follows:
   $$\mathbf{v}_k = \underset{\|\mathbf{v}\|=1}{\operatorname{argmax}} \left\| \left(B^T - \sum_{i=1}^{k-1} B^T \mathbf{v}_i \mathbf{v}_i^T \right) \mathbf{v} \right\|$$
   The coordinate transformation matrix can thus be obtained by arranging the principal components as rows of a matrix $P = [\mathbf{v}_1\, \mathbf{v}_2 ... \mathbf{v}_m]^T$.
2. Divide the link traffic space into the *normal subspace* and the *anomalous subspace*. Lakhina *et al.* [19] developed a threshold-based separation method by examining the projection of the time series of link traffic data on each principal axis in order. As soon as a projection is found that contains a 3σ deviation from the mean, that principal axis and all subsequent axes are assigned to the anomalous subspace. All previous principal axis are assigned to the normal subspace.
   We use $P_a = [\mathbf{v}_r\, \mathbf{v}_{r+1} ... \mathbf{v}_m]^T$ to denote the matrix of the principal axes in the anomalous subspace, where $\mathbf{v}_r$ is the first axis that fails to pass the threshold test.
3. The anomalous traffic can now by extracted from link load observation by first projecting the data into the anomalous subspace and then transforming it back, by taking $\tilde{B} = (P_a^T P_a)B$, and so we obtain the transformation matrix $T = P_a^T P_a$.

We call the above method *spatial PCA* because it exploits the correlation between traffic on different links (across space). Later in Section 3.3.4, we will describe *temporal PCA*, which exploits temporal correlation by applying PCA to identify dominant patterns across time.

3.3 Temporal Anomography

The anomalous link traffic can also be separated by performing temporal analysis on the time series for each link. Consider a set of link traffic data over time t: $B = [\mathbf{b}_1\, \mathbf{b}_2 ... \mathbf{b}_t]$. The process of extracting anomalies by exploiting the temporal structure within the data points can be modeled as a linear transformation of the time series: $\tilde{B} = [\tilde{\mathbf{b}}_1\, \tilde{\mathbf{b}}_2 ... \tilde{\mathbf{b}}_t] = BT$, where the transformation matrix T can be either explicit or implicit. In this paper, we consider four types of temporal analysis: ARIMA, Fourier, Wavelet, and PCA (for identifying dominant patterns across time). Although it may not be obvious at first glance, all these methods indeed fit in our framework of linear matrix transformation, as we will see next.

3.3.1 ARIMA Modeling

Univariate time series. The Box-Jenkins methodology, or AutoRegressive Integrated Moving Average (ARIMA) modeling technique [2, 3], is a class of linear time-series forecasting techniques that capture the linear dependency of the future values on the past. It is able to model a wide spectrum of time-series behavior, and has been extensively used for anomaly detection in univariate time-series.

An ARIMA model includes three order parameters: the autoregressive parameter (p), the number of differencing passes (d), and the moving average parameter (q). In the notation introduced by Box and Jenkins, models are summarized as ARIMA(p, d, q). A model described as ARIMA$(0, 1, 2)$ means that it contains $p = 0$ (zero) autoregressive parameters and $q = 2$ moving-average parameters which were computed for the time series after it was differenced once ($d = 1$).

A general ARIMA model of order (p, d, q) can be expressed as:

$$z_k - \sum_{i=1}^{p} \phi_i \cdot z_{k-i} = e_k - \sum_{j=1}^{q} \theta_j \cdot e_{k-i}, \tag{5}$$

where z_k is obtained by differencing the original time series d times (when $d \geq 1$) or by subtracting the mean from the original time series (when $d = 0$), e_k is the forecast error at time k, ϕ_i ($i = 1, ..., p$) and θ_j ($j = 1, ..., q$) are the autoregression and moving-average coefficients, respectively.

Many commonly used smoothing models are special instances of ARIMA models. For example, the Exponentially Weighted Moving Average (EWMA), is equivalent to ARIMA$(0, 1, 1)$; linear exponential smoothing, also known as Holt-Winters, is equivalent to ARIMA$(0, 2, 2)$. See [26] for detailed equations for various smoothing models and their equivalence with ARIMA models.

There are well known techniques for estimating the parameters p, d, q, ϕ_i and θ_j for a given time series [2, 3], and given the parameters, the model is simply applied to get $\hat{z}_k$ a prediction of z_k (using for instance the Durbin-Levinson algorithm [3]). The prediction errors are then $e_{k+1} = z_{k+1} - \hat{z}_{k+1}$, which then form our anomalous traffic (the traffic which does not fit the model). In practice the parameters used in the ARIMA model are sometimes chosen to meet particular goals intended by the implementor (see [4] for some discussion of these choices), rather than being estimated from the data set, because the parameters of a data set may change over time. However, we prefer to use adaptive techniques to overcome this problem.

If we consider the time series to be vectors of length t, then the above results can be written in matrix form. Taking the measurements $\mathbf{b} = (b_1, \ldots, b_t)^T$, we can obtain the errors $\mathbf{e} = (e_1, \ldots, e_t)^T$, via right-multiplication by a transformation matrix $\tilde{\mathbf{b}}^T = \mathbf{e}^T = \mathbf{b}^T T$. Specifically, let I denote the $t \times t$ identity matrix, $\bigtriangledown$ denote the "back shift" matrix, and $\mathbb{1}$ denote the $t \times t$ unit matrix, i.e.,

$$I = \begin{bmatrix} 100...00 \\ 010...00 \\ \cdots \\ 000...10 \\ 000...01 \end{bmatrix}, \bigtriangledown = \begin{bmatrix} 010...00 \\ 001...00 \\ \cdots \\ 000...01 \\ 000...00 \end{bmatrix}, \mathbb{1} = \begin{bmatrix} 111...11 \\ 111...11 \\ \cdots \\ 111...11 \\ 111...11 \end{bmatrix}.$$

The differencing result, $\mathbf{z} = [z_1 z_2 ... z_t]^T$, is then

$$\mathbf{z}^T = \begin{cases} \mathbf{b}^T (I - \bigtriangledown)^d, & \text{for } d \geq 1, \\ \mathbf{b}^T - \frac{1}{t}\mathbf{b}^T \mathbb{1} = \mathbf{b}^T \left(I - \frac{1}{t}\mathbb{1}\right), & \text{for } d = 0. \end{cases} \tag{6}$$

Equation (5) can be written in matrix notation as

$$\mathbf{z}^T - \sum_{i=1}^{p} \phi_i \mathbf{z}^T \bigtriangledown^i = \mathbf{e}^T - \sum_{j=1}^{q} \theta_j \mathbf{e}^T \bigtriangledown^j,$$

or equivalently,

$$\mathbf{e}^T = \mathbf{z}^T \left(I - \sum_{i=1}^{p} \phi_i \bigtriangledown^i\right) \left(I - \sum_{j=1}^{q} \theta_j \bigtriangledown^j\right)^{-1}.$$

Extending ARIMA based models to multivariate time series is straightforward. As noted earlier, we construct the matrix B with the measurements at each time period $\mathbf{b}_i$ as its columns. Via the above transformations, we obtain

$$E = Z \left(I - \sum_{i=1}^{p} \phi_i \bigtriangledown^i\right) \left(I - \sum_{j=1}^{q} \theta_j \bigtriangledown^j\right)^{-1}. \tag{7}$$

ARIMA based anomography. Replacing Z by the matrix form of (6), we see that $E = BT$ is indeed a transformation given by right-multiplying B with a matrix T. In fact, any *linear* filtration of the elements of a time series can be modeled by a right multiplying matrix transformation.

To get back to anomaly detection, we simply identify the forecast errors as anomalous link traffic, $\tilde{B} = E$. That is, traffic behavior that cannot be well captured by the model is considered anomalous.

3.3.2 Fourier Analysis

Fourier analysis [21] is the process of decomposing a complex periodic waveform into a set of sinusoids with different amplitudes, frequencies and phases. The sum of these sinusoids can exactly match the original waveform. This lossless transform presents a new perspective of the signal under study (in the frequency domain), which has proved useful in very many applications.

For a discrete-time signal $x_0, x_1, \ldots, x_{N-1}$, the Discrete Fourier Transform (DFT) is defined by

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{-jk2\pi n/N}, \quad \text{for } 0 \leq n \leq N-1,$$

where f_n is a complex number that captures the amplitude and phase of the signal at the n-th harmonic frequency (with base frequency $1/N$). Note that for a real signal $\{f_n\}$ is symmetric, i.e., $f_n = f_{N-1-n}$. Lower n corresponds to a lower frequency component, with f_0 being the DC component, or the average of the input series, and f_n with n close to $N/2$ corresponding to high frequencies.

The Inverse Discrete Fourier Transform (IDFT) is used to reconstruct the signal in the time domain by

$$x_n = \sum_{k=0}^{N-1} f_k e^{jk2\pi n/N}, \quad \text{for } 0 \leq n \leq N-1.$$

An efficient way to implement the DFT and IDFT is through an algorithm called the Fast Fourier Transform (FFT). The computational complexity of the FFT is $O(N \log(N))$.

FFT based anomography. The idea of using the FFT to extract anomalous link traffic, $\tilde{B}$ is to filter out the low frequency components in the link traffic time series. In general, low frequency components capture the daily and weekly traffic patterns, while high frequency components represent the sudden changes in traffic behavior. Working in the frequency domain provides us with the opportunity to distinguish these two kinds of behaviors.

We summarize FFT based anomography as follows.

1. Transform link traffic B into the frequency domain: $F = \text{FFT}(B)$: apply the FFT on each row of B. (Recall that a row corresponds to the time series of traffic data on one link.) The result is the corresponding frequency domain series, in each row of F.
2. Remove low frequency components: i.e. set $F_i = 0$, for $i \in [1, c] \cup [N - c, N]$, where F_i is the i-th column of F and c is a cut-off frequency. (For example, for the results presented in Section 6, we use 10-minute aggregated link traffic data of one week duration, and $c = \lceil \frac{10}{60} N \rceil$, corresponding to a frequency of one cycle per hour.)
3. Transform back into the time domain: i.e. we take $\tilde{B} = \text{IFFT}(F)$. The result is the high frequency components in the traffic data, which we will use as anomalous link traffic, $\tilde{B}$.

The DFT and IDFT may be represented as right-matrix products. In setting columns of F to zero, and performing the IDFT we are taking a linear combination of the columns of F, which in turn are a linear combination of those of B. Hence, the overall process above can be modeled as a right-multiplying matrix transformation $\tilde{B} = BT$. Note also that in thresholding at frequency c we preserve the symmetry of F, and so although F may contain complex elements, the resulting transform will be real.

3.3.3 Wavelet Analysis

Wavelets [7, 12, 21] are mathematical functions that cut up data into different frequency components, and then study each component with a resolution matched to its scale. They provide a powerful means for isolating characteristics of signals via a combined time-frequency representation and are often considered superior to traditional Fourier methods especially in situations where the signal contains transients, such as discontinuities and sharp spikes.

In [1], Barford *et al.* have developed a wavelet-based algorithm for detecting anomalies in the link traffic data. It shares the same principle as the FFT based approaches — exposing anomalies by filtering low frequency components. More specifically, it uses wavelets to decompose the original signal into low-, mid-, and high-frequency components and then detects anomalies by close examination of the mid- and high-frequency components.

Below we compute $\tilde{B}$ as the high-frequency components of link traffic B. We can also compute $\tilde{B}$ as the mid-frequency components of B in essentially the same way.

1. Use wavelets to decompose B into different frequency levels: $W = \text{WAVEDEC}(B)$, by applying a multi-level 1-D wavelet decomposition on each row of B. The result is a wavelet decomposition vector, which we save as one row in matrix W. The wavelet we use is the Daubechies wavelet [6] of order 6.
2. Then remove low- and mid-frequency components in W by setting all coefficients at frequency levels higher than w_c to 0. Here w_c is a cut-off frequency level. For the results presented in Section 6, we use 10-minute aggregated link traffic data of one week duration, and w_c is set at 3. That is, we only keep coefficients at frequency levels 1, 2, and 3, which is consistent with [1].
3. Reconstruct the signal: $\tilde{B} = \text{WAVEREC}(B)$. The result is the high-frequency components in the traffic data.

It is easy to verify that the process of WAVEDEC and WAVEREC only involves linear combinations of columns of B. As a result, the $\tilde{B}$ derived through the wavelet based anomography can also be modeled as right multiplying matrix transformation.

3.3.4 Temporal PCA

In Section 3.2.1, we presented a method of applying PCA to find dominant patterns among different link-load time series. A similar method can be used in identifying dominant patterns across time.

Consider the link load matrix $B = [\mathbf{b}_1 \, \mathbf{b}_2 ... \mathbf{b}_t]$. We can think of each row as a t-dimensional vector. What we are looking for is a new coordinate system, $\mathbf{v}_1$, $\mathbf{v}_2$, ... ,$\mathbf{v}_t$, such that the projection of the m links (on $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_t$) has energy concentrated on the first several axes. This is exactly what PCA provides. The only difference is that we now apply PCA on B^T as opposed to B (as used in spatial PCA). Then we follow the same procedure to define an anomalous subspace and to extract anomalies that have projections in the anomalous subspace. In this way, we obtain a left multiplying transformation matrix T, i.e., $\tilde{B}^T = TB^T$. Taking transpose on both side of the equation, we have $\tilde{B} = (\tilde{B}^T)^T = (TB^T)^T = BT^T$ where T^T is a right multiplying transformation matrix that extracts anomalies from B.

3.4 Inference Algorithms

Once we obtain the matrix of link anomalies $\tilde{B}$, the next step is to reconstruct OD flow anomalies $\tilde{X}$ by solving a series of ill-posed linear inverse problems $\tilde{\mathbf{b}}_j = A\tilde{\mathbf{x}}_j$. For example, Lakhina *et al* [19] proposed to find the single largest anomaly in each time interval j by applying a greedy algorithm. We present below three common inference algorithms for solving these problems. All three algorithms deal with the underconstrained linear system by searching for a solution that minimizes some notions of vector norm, three examples of which are

- The ℓ^2 norm of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\|_2 = \left(\sum_i v_i^2\right)^{\frac{1}{2}}$, where v_i is the i-th element of vector $\mathbf{v}$.
- The ℓ^1 norm of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\|_1 = \sum_i |v_i|$, *i.e.*, the sum of the absolute value of each element of $\mathbf{v}$.
- The ℓ^0 norm of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\|_0 = \sum_i v_i^0$, *i.e.*, the number of non-zero elements of $\mathbf{v}$.

3.4.1 Pseudoinverse Solution

A standard solution to $\tilde{\mathbf{b}} = A\tilde{\mathbf{x}}$ is the pseudoinverse solution $\tilde{\mathbf{x}} = A^+\tilde{\mathbf{b}}$, where A^+ is the pseudoinverse (or Moore-Penrose inverse) of matrix A. It is known that $\tilde{\mathbf{x}} = A^+\tilde{\mathbf{b}}$ is the solution to the problem $\tilde{\mathbf{b}} = A\tilde{\mathbf{x}}$ that minimizes the ℓ^2 norm of the anomaly vector, i.e. it solves:

$$\text{minimize } \|\tilde{\mathbf{x}}\|_2 \quad \text{subject to } \|\tilde{\mathbf{b}} - A\tilde{\mathbf{x}}\|_2 \text{ is minimal.} \tag{8}$$

3.4.2 Sparsity Maximization

In practice, we expect only a few anomalies at any one time, so $\tilde{\mathbf{x}}$ typically has only a small number of large values. Hence it is natural to proceed by maximizing the *sparsity* of $\tilde{\mathbf{x}}$, *i.e.*, solving the following ℓ^0 norm minimization problem:

$$\text{minimize } \|\tilde{\mathbf{x}}\|_0 \quad \text{subject to } \tilde{\mathbf{b}} = A\tilde{\mathbf{x}}. \tag{9}$$

The ℓ^0 norm is not convex and is notoriously difficult to minimize, so in practice one needs to either approximate the ℓ^0 norm with a convex function or use heuristics, for example the greedy algorithm of Lakhina *et al* [19].

ℓ^1 norm minimization One common approach to approximate ℓ^0 norm minimization is to convexify (9) by replacing the ℓ^0 norm with an ℓ^1 norm, so that we seek a solution to

$$\text{minimize } \|\tilde{\mathbf{x}}\|_1 \quad \text{subject to } \tilde{\mathbf{b}} = A\tilde{\mathbf{x}} \tag{10}$$

As shown in [8, 9], ℓ^1 norm minimization results in the sparsest solution for many large under-determined linear systems.

In the presence of measurement noise, the constraints $\tilde{\mathbf{b}} = A\tilde{\mathbf{x}}$ may not always be satisfiable. In this case, we can add a penalty term $\|\tilde{\mathbf{b}} - A\tilde{\mathbf{x}}\|_1$ to the objective and reformulate (10) as:

$$\text{minimize} \quad \lambda\|\tilde{\mathbf{x}}\|_1 + \|\tilde{\mathbf{b}} - A\tilde{\mathbf{x}}\|_1 \tag{11}$$

where $\lambda \in [0, 1]$ controls the degree to which the constraints $\tilde{\mathbf{b}} = A\tilde{\mathbf{x}}$ are satisfied. As shown in Section 6, the algorithm is not very sensitive to the choice of λ. In the rest of this paper, unless noted otherwise, we use $\lambda = 0.001$, which gives satisfactory results.

We can cast (11) into the following equivalent Linear Programming (LP) problem, for which solutions are available even when A is very large, owing to modern interior-point linear programming methods.

$$\begin{array}{ll} \text{minimize} & \lambda \sum_i u_i + \sum_j v_j \\ \text{subject to} & \tilde{\mathbf{b}} = A\tilde{\mathbf{x}} + \mathbf{z} \\ & \mathbf{u} \geq \tilde{\mathbf{x}}, \quad \mathbf{u} \geq -\tilde{\mathbf{x}} \\ & \mathbf{v} \geq \mathbf{z}, \quad \mathbf{v} \geq -\mathbf{z} \end{array} \tag{12}$$

Greedy algorithm Another common heuristic solution for ℓ^0 norm minimization is to apply the greedy algorithm. For example, the greedy heuristic has been successfully applied to wavelet decomposition, where it goes by the name of *Orthogonal Matching Pursuit* (OMP) [24]. In the same spirit here, we develop a greedy solution to maximize the sparsity of $\tilde{\mathbf{x}}$. The algorithm starts with an empty set I of non-zero positions for $\tilde{\mathbf{x}}$ and then iteratively adds new non-zero positions to I. During each iteration, for each position $p \notin I$, the algorithm tests how much it can reduce the residual $\tilde{\mathbf{b}} - A\tilde{\mathbf{x}}$ by including p as a non-zero position. More specifically, let $J = I \cup \{p\}$. The algorithm estimates the values for the non-zero elements of $\tilde{\mathbf{x}}$ (denoted as $\tilde{\mathbf{x}}_J$) by solving the following least squares problem

$$\text{minimize } \|\tilde{\mathbf{b}} - A_J\tilde{\mathbf{x}}_J\|_2 \tag{13}$$

where $A_J = A[., J]$ is a submatrix of A formed by the column vectors of A corresponding to positions in J. The residual is then computed as $e_J = \|\tilde{\mathbf{b}} - A_J\tilde{\mathbf{x}}_J\|_2$. The algorithm then greedily chooses the position p that gives the smallest e_J and adds it to I. The algorithm stops whenever either the residual energy falls below some tolerance to inaccuracy $e_{\max}$ or the number of non-zero positions exceeds some threshold $\ell^0_{\max}$.

4 Dynamic Network Anomography

Up to this point, we have assumed that the routing matrices are constant. However, we wish to allow for dynamic routing changes, and so we must allow A_j to vary over time. In IP networks, routing changes occur as part of the normal "self-healing" behavior of the network, and so it is advantageous to isolate these from traffic anomalies and only signal traffic anomalies. In addition, if some measurements are missing (say at time j), we may still form a consistent problem by setting the appropriate rows of A_j to zero. Thus, for realistic SNMP measurements where missing data are often an issue, we still wish to vary A_j even for static routing. Routing measurements may be obtained using a route monitor, to provide accurate, up-to-date measurements of routing (at least at the time scale of SNMP measurements, e.g. minutes).

Where the tomography step can be done separately at each time interval (for instance see [34, 35]), it is simple to adapt early-inverse methods to *dynamic network anomography* by inverting (2) at each time step. Given the straight forward approach for early-inverse methods, We seek here to generalize late-inverse methods to dynamic network anomography.

4.1 Dynamic Temporal Anomography

When the routing matrix is non-constant, there is no reason to believe that the measurements B should follow a simple model such as an ARIMA model. Even where the traffic itself follows such a model, a simple routing change may change a link load measurement by 100%, for instance by routing traffic completely away from a particular link. If we were to apply the ARIMA model to the measurements B, we would see such a change in routing as a level-shift anomaly. However, its cause is not an unknown change in X (to be discovered), but rather a known change in the routing matrices A_j. Likewise, it no longer makes sense to try to exploit spatial correlations which arose from a particular routing, to the case of another routing.

However, it is no less reasonable to approximate the traffic matrix X by an ARIMA model (than B when the routing is constant), even when routing may change. Under such a modeling assumption, we can write $\tilde{X} = XT$. We know also that the measurements are given by (2). A reasonable approach to the solution is therefore to seek a solution $\tilde{X}$ which is consistent with these equations, but also minimizes one of the norms (described above) at each time step. We choose to minimize the ℓ^1 norm $\|\tilde{\mathbf{x}}_j\|_1$ here because (i) it allows us to naturally incorporate link load constraints at multiple time intervals, and (ii) it is more accurate than both the pseudoinverse and the greedy algorithms for static anomography (as we will show in Section 6).

Unfortunately, for transform based methods (the Fourier, wavelet and PCA methods) the number of constraints becomes very large (as t grows). On the other hand, the set of constraints for the ARIMA model can be written in a form such that it does not grow with t. Hence, in the following we concentrate on generalizing the ARIMA approach. We present the algorithm for ARIMA(p, d, q) models with $d \geq 1$ (Section 4.2). We have also extended the algorithm to handle ARIMA models with $d = 0$, though we omit this treatment here for brevity (as it is a straightforward extension). Due to space limits, we will leave out the discussion on model selection and parameter estimation, two important issues for applying ARIMA-based anomography. Interested readers can find this in our technical report [33].

4.2 Algorithm for ARIMA Models ($d \geq 1$)

We are going to seek solutions that are consistent with the measurements $\mathbf{b}_j = A_j\mathbf{x}_j$, for $j = 1, \ldots, t$, and an ARIMA model that gives $\tilde{X} = XT$ where T is the same transformation matrix implicitly defined by (6) and (7). Importantly, we do not wish to have to estimate X (or we may as well use an early-inverse method). The advantage of the ARIMA model, is we do not need to know X, but only linear combinations of X.

Let L be the backshift operator, whose effect on a process $\mathbf{z}$ can be summarized as $(L\mathbf{z})_k = \mathbf{z}_{k-1}$. Let the AR polynomial $\Phi(L)$ be

$$\Phi(L) = \sum_{i=0}^{d+p} \gamma_i L^i \stackrel{\text{def}}{=} \left(1 - \sum_{i=1}^{p} \phi_i L^i\right)(1-L)^d.$$

Let $\mathbf{y}_{k-i} = \gamma_i \mathbf{x}_{k-i}$. We now identify $\mathbf{e} = \tilde{\mathbf{x}}$ in the ARIMA model described in (5) (or rather its multivariate extension). By definition the sum $\sum_{i=0}^{d+p} \mathbf{y}_{k-i} = \mathbf{z}_k - \sum_{i=1}^{p} \phi_i \mathbf{z}_{k-i}$, and so, for $d \geq 1$, the ARIMA model (5) can be rewritten

$$\sum_{i=0}^{d+p} \mathbf{y}_{k-i} = \tilde{\mathbf{x}}_k - \sum_{j=1}^{q} \theta_j \tilde{\mathbf{x}}_{k-j}. \qquad (14)$$

Define $\mathbf{c}_{k-i} = \gamma_i \mathbf{b}_{k-i}$, then as $\mathbf{y}_{k-i} = \gamma_i \mathbf{x}_{k-i}$, the measurement equation (2) implies

$$A_{k-i}\mathbf{y}_{k-i} = \mathbf{c}_{k-i}, \quad i = 0, 1, \cdots, d+p. \qquad (15)$$

We can compute $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \cdots, \tilde{\mathbf{x}}_t$ iteratively by solving a series of ℓ^1 norm minimization problems $\mathcal{P}_k$ ($k = 1, 2, \cdots, t$):

$$\mathcal{P}_k: \quad \text{minimize } \|\tilde{\mathbf{x}}_k\|_1 \text{ subject to (14) and (15).} \qquad (16)$$

As an example, consider the simplest ARIMA model, ARIMA$(0, 1, 0)$. In this case, $p = q = 0$, so we have

$$\Phi(L) = \sum_{i=0}^{1} \gamma_i L^i = (1 - L),$$

so $\gamma_0 = 1$ and $\gamma_1 = -1$, and (14) becomes $\tilde{\mathbf{x}}_k = \sum_{i=0}^{1} \mathbf{y}_{k-i}$, thus problem $\mathcal{P}_k$ is simply

$$\begin{array}{ll} \text{minimize} & \|\tilde{\mathbf{x}}_k\|_1 \\ \text{subject to} & \left\{ \begin{array}{rcl} \tilde{\mathbf{x}}_k & = & \mathbf{y}_k + \mathbf{y}_{k-1} \\ A_k\mathbf{y}_k & = & \mathbf{b}_k \\ A_{k-1}\mathbf{y}_{k-1} & = & -\mathbf{b}_{k-1} \end{array} \right. \end{array} \qquad (17)$$

As in Section 3.4.2, we can accommodate measurement noise by incorporating penalty terms into the objective to penalize against violation of constraints (14) and (15). We can then solve the resulting ℓ^1 norm minimization problem by reformulating it as an equivalent LP problem. We omit such details in the interest of brevity.

We have also developed two techniques to significantly reduce the size of the above minimization problems $\mathcal{P}_k$ by exploiting the fact that changes in routing matrices tend to be infrequent (*i.e.*, not in every time interval) and local (*i.e.*, only in a small subset of rows). Interested readers please refer to our technical report [33] for details.

5 Evaluation Methodology

5.1 Data Sets

We apply our techniques to real network measurement data gathered from two large backbone networks – Internet2's Abilene network and a Tier-1 ISP network. Both networks span the continental USA. The Abilene backbone is relatively small, with 12 core routers, 15 backbone links and 144 OD flow elements in its traffic matrix. The Tier-1 ISP network is much larger, consisting of hundreds of routers, thousands of links and tens of thousands of different OD flows. To reduce computation complexity without loss of utility, we use the technique in [34] to lump edge routers with topologically equivalent connectivity. This reduces the total number of OD flows to about 6000.

The primary data inputs for our anomaly diagnosis are the time series of link loads (bytes across interfaces) for every network, gathered through SNMP. We use flow level data, where available, for validation. As is often the case, the flow data is incomplete. The flow data are collected at the edge of the network where data packets are sampled and aggregated by the IP source and destination address, and the TCP port number. Adjusted for sampling rate and combined with BGP and ISIS/OSPF routing information, these sampled IP flow statistics are then aggregated into a real traffic matrix [11], where each element is an OD flow with the origin and destination being the ingress and egress point of the flow to/from the network. Consistent with [19], we aggregate these measurements into bins of 10 minutes to avoid any synchronization issues that could have arisen in the data collection.

Ideally, to evaluate the methods, one would like complete flow level data, SNMP link load measurements, and continuous tracking of routing information, providing a consistent, comprehensive view of the network in operation. Unfortunately, we do not have the complete set of flow level data across the edge of the network (due to problems in vendor implementations or in data collection), and our routing information is only "quasi-" real time (we rely on snapshots available from table dumps carried out every 8 hours). As a result, inconsistencies sometimes arise between these measurements. To overcome these problems and provide a consistent means for evaluating the algorithms, we adopt the method in [34] and reconstruct the link traffic data by simulating the network routing on the OD flow traffic matrix generated from the available set of flow level data. Note that we use derived link load measurements for validation purposes only. In practice, our methods are applicable to direct measurement of traffic data as obtained from SNMP.

5.2 Performance Metrics

We conduct our evaluation in two steps. First, we compare the different solution techniques for the inverse problem $\tilde{\mathbf{b}}_j = A\tilde{\mathbf{x}}_j$ (as described in Section 3.4). The inverse problem is common to all the late-inverse anomography methods discussed in Section 3, so for simplicity we choose to use the simplest temporal forecasting model, ARIMA$(0, 1, 0)$, for evaluation. This model predicts the next observation to have the same value as the current one. Thus, the inverse problem on the prediction error can be constructed by simply taking the difference between consecutive link load observations: $A\tilde{\mathbf{x}}_t = \tilde{\mathbf{b}}_t = \mathbf{b}_t - \mathbf{b}_{t-1}$. The performance of the inversion technique is measured by comparing the inferred solution, $\tilde{\mathbf{x}}_t$, to the direct difference of the OD flow, $\mathbf{x}_t - \mathbf{x}_{t-1}$; the closer the values are, the better the result. In the context of anomaly detection, it is often the case that the large elements (large volume changes) are of chief interest to network management. Hence, we defined a metric – detection rate – to compare the top ranked N elements (sorted by size) in solution $\tilde{\mathbf{x}}_t$ to the top N prediction errors $\mathbf{x}_t - \mathbf{x}_{t-1}$ for t spanning a period of one week. As we will see in Section 6, the top anomalies in our data are easily resolved by magnitude (close ties are rare). The *detection rate* is the ratio of the overlap between the two sets. Note that the detection rate avoids some problems with comparing false-alarm versus detection probabilities, as it combines both into one measure. A high detection rate indicates good performance. Detection rate is used to compare inference techniques in Section 6.1, to assess sensitivity to λ, robustness to noise in Section 6.2, and the effectiveness of the methods for time-varying routing in Section 6.3.

In Section 6.4.2 we step away from the simple anomaly detection algorithm applied to test the inference component, and compare the complete set of anomography methods described in Section 3. As before we use detection rate to measure whether the anomaly detection method produces similar results when applied to the OD pairs directly, or applied to the link load data, along with an inversion method — we use the Sparsity-L1 method (the best performing of the methods tested using the methodology above). In other words, we benchmark the anomography method against the anomalies seen in direct analysis of the OD flows.

Since different methods may find different sets of benchmark anomalies, we need an objective measure for assessing the performance of the methods. Ideally, we would like to compare the set of anomalies identified by each of the methods to the set of "true" network anomalies. However, isolating and verifying all genuine anomalies in an operational network is, although important, a very difficult task. It involves correlating traffic changes with other data sources (e.g., BGP/OSPF routing events, network alarms, and operator logs), an activity that often involves case-by-case analysis. Instead, we perform pair-wise comparisons, based on the top ranked anomalies identified by each of the anomography methods, an approach also taken in Lakhina *et al.* [19].

Specifically, for each of the anomography methods, we apply the underlying anomaly detection method directly to the OD flow data. We think of the top ranked M anomalies, denoted by the set $\mathcal{B}_M^{(j)}$ for anomaly detection method j as a benchmark. For each of the anomography methods i, we examine the set of N largest anomalies $\mathcal{A}_N^{(i)}$ inferred from link load data. To help understand the fidelity of the anomography methods we consider the overlap between the benchmark and the anomography method, $\mathcal{A}_N^{(i)} \cap \mathcal{B}_M^{(j)}$, across the benchmarks and the anomography methods. We allow a small amount of slack (within one ten-minute time shift) in the comparison between events, in order that phase differences between methods not unduly impact the results.

We are interested in understanding both false positives and false negatives:

(i) False Positives. Taking $\mathcal{B}_M^{(j)}$ as the benchmark, the false positives produced by anomography method i are $\mathcal{A}_N^{(i)} - \mathcal{B}_M^{(j)}$. The magnitudes of the anomalies in $\mathcal{A}_N^{(i)}$ and $\mathcal{B}_M^{(j)}$ may vary. Yet, intuitively if one of the $N = 30$ top anomalies in $\mathcal{A}_N^{(i)}$ is not among the top $M = 50$ from the benchmark, then this anomaly in $\mathcal{A}_N^{(i)}$ is likely a false positive. This leads to the following heuristic for detecting false positives. We choose (reasonable) parameters N and M, with $N < M$, and count the false positives as the size of $\mathcal{A}_N^{(i)} - \mathcal{B}_M^{(j)}$.

(ii) False Negatives. Our reasoning is similar. Taking $\mathcal{B}_M^{(j)}$ as the benchmark, the false negatives produced by anomography method i are $\mathcal{B}_M^{(j)} - \mathcal{A}_N^{(i)}$. Intuitively if one of the $M = 30$ top anomalies in the benchmark is not among the top $N = 50$ anomalies in $\mathcal{A}_N^{(i)}$ then this anomaly in $\mathcal{B}_M^{(j)}$ is missed by the anomography method i, and is a false negative. This leads to the following heuristic for detecting false negatives. We choose (reasonable) parameters N and M, with $N > M$, and count the false negatives as the size of $\mathcal{B}_M^{(j)} - \mathcal{A}_N^{(i)}$.

For our reports in the next section, we choose the smaller of M and N to be 30, since this roughly represents the number of traffic anomalies that network engineers might have the resources to analyze deeply on a weekly basis. We would like to show comparative results where the larger parameter varies, but cannot within a reasonable amount of space, and so show results for one fixed value 50. It is important to note that the results we obtained for other values of M and N change none of our qualitative conclusions.

6 Results

We obtained six months (03/01/04-09/04/04) of measurements for the Abilene network and one month (10/06/04-11/02/04) for the Tier-1 ISP network. We partitioned the data into sets spanning one week each, and evaluated the methods on each data set. Due to space limits, we present only one set of representative results – Tier-1 ISP (10/6/04-10/12/04). In our technical report [33], we also report results in other weeks for the Tier-1 ISP network as well as for the Abilene network. These results are qualitatively similar to those reported here.

6.1 Comparison of Inference Techniques

We first compare different solution techniques for the inference problem $\tilde{\mathbf{b}} = A\tilde{\mathbf{x}}$. More specifically, we consider three late inverse algorithms: **Pseudoinverse** (Section 3.4.1), **Sparsity-Greedy** (Section 3.4.2), and **Sparsity-L1** (Section 3.4.2), and one early inverse technique: **Early Inverse-Tomogravity**. We choose to use the tomogravity method [35] as the early inverse technique since it has demonstrated high accuracy and robustness for estimating traffic matrix for real operational networks [14, 35].

Figure 1 plots the sizes of the top 50 anomalies (the forecast errors) of the OD flows (the solid lines) and the corresponding values diagnosed by the different inference techniques (the points) for 10/6/04 to 10/12/04, for the Tier-1 ISP network. The y-axis provides the size of the anomalies normalized by the average total traffic volume on the network. The x-axis is the rank by the size of anomalies directly computed from the OD flows. We observe that there are very few large changes – among more than 6 million elements ($\sim$ 6000 OD flows at 1007 data points), there is one instance where the size of anomaly is more than 1% of total traffic and there are 18 cases where the disturbances

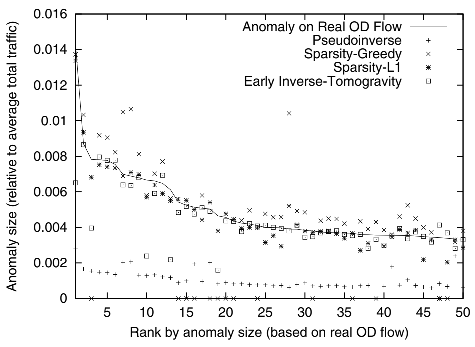


Figure 1: Anomalies by Size

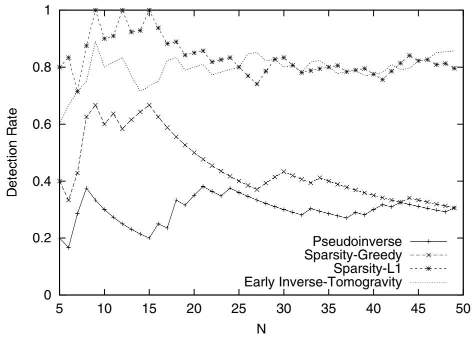


Figure 2: Detection Rate by Various Inference Techniques

constitute more than 0.5% of total traffic. This agrees with our intuition on the sparsity of network anomalies.

We see that Pseudoinverse significantly underestimates the size of the anomalies. Intuitively, Pseudoinverse finds the least square solution which distributes the "energy" of the anomaly evenly to all candidate flows that may have contributed to the anomaly, under the link load constraint. This is directly opposed to the sparsity maximization philosophy. Among the sparsity maximization techniques, Sparsity-L1 performs the best. Sparsity-L1 always finds solutions close to the real anomalies. Sparsity-Greedy, in general, is more effective than Pseudoinverse, although it sometimes overestimates the size of anomalies. As a representative of the early inverse technique, Tomogravity also performs well. With few exceptions, tomogravity finds solutions that track the real OD flow anomalies. Intuitively, when a proportionality condition holds, i.e., when the size of the anomalies are proportional to the sizes of the OD flows, then early inverse methods work well. However, where the proportionality condition does not hold, the error can be significant.

Figure 2 presents the detection rate for the different inference techniques. We observe that for the Tier-1 ISP network, Sparsity-L1 and Tomogravity, which have about 0.8 detection rate, significantly outperform other methods.

Due to space limits, we will consider only Sparsity-L1 and Tomogravity in the rest of the evaluation, as these method demonstrate the greatest performance and flexibility in dealing with problems such as missing data and routing changes.

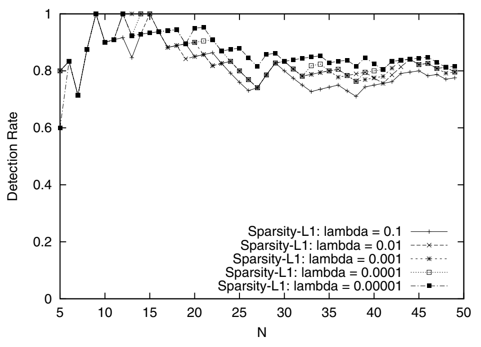


Figure 3: Sensitivity to Parameter Choice: λ

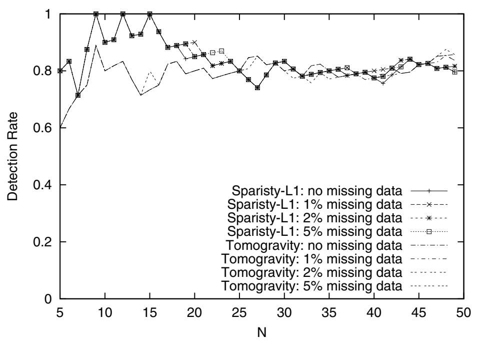


Figure 5: Impact of Missing Data

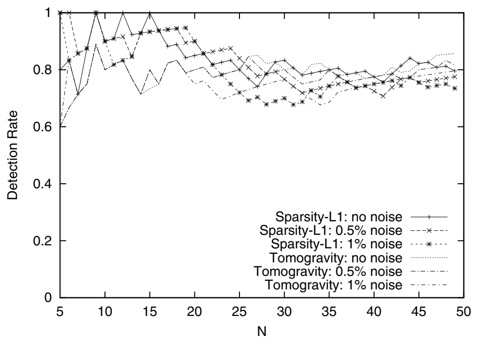


Figure 4: Sensitivity to Measurement Noise

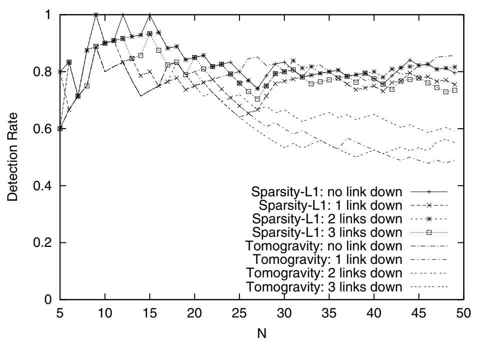


Figure 6: Impact of Route Change

6.2 Robustness

λ in Sparsity-L1. Sparsity-L1 involves a parameter λ in its formulation (Eq. 11). Figure 3 investigates the sensitivity to the parameter choice. Specifically, Figure 3 plots the detection rate of Sparsity-L1 for $\lambda = 0.1$, 0.01, 0.001, 0.0001 and 0.00001. All λ in this range achieve good performance. This is reassuring, since it suggests that little training or parameter tuning is needed to match the method to a different network or traffic pattern.

Measurement Noise. Thus far, we have assumed perfect link load information for anomaly detection. However, in real networks, SNMP byte counts are collected from all routers across the network. Inevitably, measurement issues such as lack of time synchronization may introduce noise. Below we evaluate the impact of measurement noise by multiplying white noise terms $N(1, \sigma)$ with each element of the link load, and then using the result as input to our inference algorithms.

Figure 4 compares how well the methods perform with no noise, to how well they do with noise levels $\sigma = 0.5\%$ and $\sigma = 1\%$. Note that measurement errors near 1% throughout the network are quite significant, since the size of the largest anomalies are themselves near 1% of the total traffic (Figure 1). It is a challenging task to accurately diagnose anomalies given the comparable level of noise. Nevertheless, we find that both Sparsity-L1 and Tomogravity are quite robust to measurement noise. For the Tier-1 ISP network, the detection rate remains above 0.8 for big anomalies (small N) and above 0.7 for the top 50 anomalies. These results demonstrate the strength of our algorithms in dealing with imperfect measurements.

6.3 Time Varying Routing Matrices

Missing Data. Missing measurement data, arising from problems such as packet loss during data collection, is common in real networks. Indeed, this can be tricky to deal with, since the loss of link load data has the effect of producing time varying routing matrices in the anomography formulation. Fortunately, as discussed in Section 4, our extended Sparsity-L1 algorithm can handle this situation.

Figure 5 shows the performance of the inference algorithms with up to 5% of the data missing – missing values are selected uniformly at random. We see that both Sparsity-L1 and Tomogravity suffer only minor (almost negligible) performance impact, in terms of detection rate. The low sensitivity to missing data is an important feature of these methods, which is critical for real implementation.

Routing Changes. In an operational network, the routing matrix is unlikely to remain unchanged over a few days. Hardware failures, engineering operations, maintenance and upgrades all may cause traffic to be rerouted on alternative paths. Here we evaluate the impact of routing changes on the performance of our algorithms. We introduce routing changes by simulating faults on internal links.

Figure 6 presents results where we have randomly failed/repaired up to 3 links at each time instance. We observe that Sparsity-L1 is very robust to such a disturbance in the routing structure, while Tomogravity suffers significant performance impact. It appears that Tomogravity suffers here because errors in the (early) inference step, being computed from different routing matrices, add to become comparable to the anomalies themselves. This demonstrates another advantage of the late-inverse over the early-inverse approach.

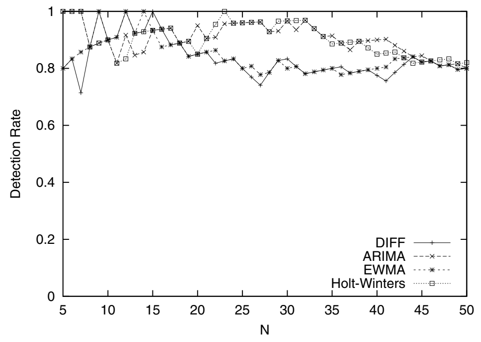
6.4 Comparison of Anomography Methods

6.4.1 Impacts on Inference Accuracy

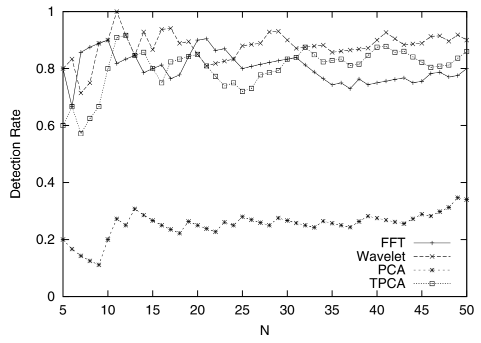
Thus far, we have compared the performance of Sparsity-L1 and Early Inverse-Tomogravity, under the simple temporal model (forecasting the next data point using the current value). We found that Sparsity-L1 in general outperforms the Early Inverse approach. We also observed that Sparsity-L1 is robust to measurement noise, is insensitive to parameter choice, and is able to handle missing data and route changes. We now evaluate overall performance when applying Sparsity-L1 with other temporal and spatial anomography methods. In particular, we compare **FFT** (Section 3.3.2), **Wavelet** (Section 3.3.3), **PCA** (Section 3.2.1), **TPCA** (Section 3.3.4), and four ARIMA based methods, **Diff** (the simple forecasting model of the last section), **Holt-Winters**, **EWMA**, and general **ARIMA**, which determines the appropriate model using the method in [33].

As noted in Section 5, for each model considered, we compute $\tilde{\mathbf{x}}$ directly from the OD flow traffic data and use it as the benchmark. Next, we compute $\tilde{\mathbf{b}}$ with the same anomography model, and construct the $A\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ inference problem. We compare the solution derived through Sparsity-L1 with the benchmark. Figure 7 presents the detection rate for these approaches. To avoid overcrowding the graph, we divide the anomography methods into two groups. Figure 7 (a) plots the results for the ARIMA family of anomography approaches and Figure 7 (b) plots the results for the rest. We observe that for all the ARIMA based approaches, Sparsity-L1 finds very good solutions. With the traffic data aggregated at the 10-minute level, simple Diff and EWMA can sufficiently extract the anomalous traffic and warrant a solution that maximizes the sparsity of the anomalies. Holt-Winters produces better performance than Diff and EWMA. This is because the model is more sophisticated, and thus is able to capture more complex temporal trends exhibited in the traffic data. Further sophistication, as incorporated in ARIMA, however, cannot significantly improve performance. In the family of ARIMA models, Holt-Winters appears to provide the best complexity-performance trade-off.

From Figure 7 (b), we observe that Sparsity-L1 can also achieve high detection rate under FFT, Wavelet and TPCA. However, it doesn't work well with PCA[2]. This can be explained as follows. When we apply spatial PCA on the real traffic matrix X and the link load matrix B, we obtain two linear transformation $\tilde{X} = T_x X$, and $\tilde{B} = T_b B = T_b A X$, respectively. However, the two transformation matrices T_x and T_b may differ significantly because the spatial correlation among link loads and that among OD flows are rather different. Even if we use $T_x = T_b$, we cannot ensure that $AT_x X = T_b AX$ (i.e., $A\tilde{X} = \tilde{B}$ (Note that this



(a) ARIMA family of anomography methods



(b) Other anomography methods

Figure 7: Sparsity-L1 with Various Anomography Methods

last comment applies to spatial anomography methods in general). Thus, the spatial PCA anomography solution is not expected to completely overlap with the $\tilde{\mathbf{x}}$ identified by directly applying spatial PCA on the OD traffic flows. In contrast, the temporal anomography methods are *self-consistent* in that given $\tilde{B} = BT$, if we apply the same transformation T on X and obtain $\tilde{X} = XT$, we guarantee that $\tilde{B} = A\tilde{X}$ $(= AXT)$.

6.4.2 Cross Validation for Different Methods

We now turn to comparing the various anomography methods . To do so, we use a set of benchmarks, as described in Section 5, each derived from applying anomaly detection algorithm directly to the OD flows. For each benchmark, we report on the success of all of the anomography methods. The hope is that methods emerge that achieve both low false positives and low false negatives for nearly all of the benchmarks.

In Table 1 (a) we present the false positives for the Tier-1 ISP network with $M = 50$ and $N = 30$ (see Section 5). We found results for different values of M and N to be qualitatively quite similar. To align our results with the methodology reported in [19], we include the bottom row, labeled PCA*, where we use a squared prediction error (SPE) based scheme to determine the set of time intervals at which big anomalies occur, and the greedy approach (Section 3.4.2) to solve the inference problem. Note that the number of anomalies reported by PCA* may be less than

Top 30 Inferred	False Positives with Top 50 Benchmark							
	Diff	ARIMA	EWMA	H-W	FFT	Wavelet	PCA	TPCA
Diff	**3**	**6**	**3**	**6**	**6**	**4**	14	14
ARIMA	**4**	**1**	**4**	**1**	**8**	**3**	10	13
EWMA	**3**	**6**	**3**	**6**	**7**	**5**	15	13
Holt-Winters	**4**	**1**	**4**	**1**	**8**	**3**	10	13
FFT	**6**	**6**	**6**	**7**	**2**	**6**	18	19
Wavelet	**6**	**6**	**6**	**6**	**8**	**1**	12	13
TPCA	17	17	17	17	20	13	14	0
PCA	18	18	18	18	20	14	14	1
PCA*(37)	18	17	18	17	23	16	11	8

Table 1: False positives seen in the top 30 inferred anomalies compared against the top 50 benchmark anomalies.

Top 50 Inferred	False Negatives with Top 30 Benchmark							
	Diff	ARIMA	EWMA	H-W	FFT	Wavelet	PCA	TPCA
Diff	0	1	0	1	5	5	12	17
ARIMA	1	0	1	0	6	4	12	18
EWMA	0	1	0	1	5	5	12	17
Holt-Winters	1	0	1	0	6	4	12	18
FFT	3	8	4	8	1	7	18	19
Wavelet	0	2	1	2	5	0	11	13
TPCA	14	14	14	14	19	15	15	3
PCA	10	13	10	13	15	11	13	1
PCA*(37)	17	18	18	18	21	19	16	8

Table 2: False negatives seen in the top 50 inferred anomalies compared against the top 30 benchmark anomalies.

N. We therefore report the actual number of anomalies in the table next to the label PCA*.

From the table, we observe from the upper left 6×6 quadrant that the ARIMA, FFT and Wavelet approaches tend to have relative low false positives among detected anomalies. Thus, the top 30 ranked anomalies derived through these approaches indeed appear to be anomalous traffic events that are worth investigating.

The PCA based approaches, however, exhibit a higher false positives when benchmarked against other approaches. This appears to be partially due to PCA identifying anomalies of a different type than those identified by the methods. Consider, for example, a sudden increase of traffic for an OD flow that persists for a couple of hours. PCA methods may identify every instance within the two-hour period as anomalous. ARIMA based approaches detect abrupt traffic changes. Hence ARIMA based methods likely extract only the "edges" – the first and last instance – of the two-hour period. Another factor contributing to PCA's false positives may be its lack of self-consistency: anomalies present in the OD pairs but not detected by the method in the link loads. In addition, unlike ARIMA, FFT, or wavelet based tomography, both spatial PCA and temporal PCA cannot fully utilize temporal ordering information in the measured time series data. For example, any reordering of the time series, $\mathbf{b}_1$, $\mathbf{b}_2$, ...,$\mathbf{b}_t$, does not affect the outcome of the algorithm.

Table 2 presents the number of false negatives for $M = 30$ and $N = 50$, where we are interested in the number of large anomalies that are not identified by each approach. We observe that the ARIMA methods, FFT and Wavelet anomography approaches have superb performance – the number of false negatives are very low. This indicates that very few important traffic anomalies can pass undetected by these approaches. The PCA based approaches, however, identify about half of the anomalies.

7 Conclusions

In this paper, we introduced *network anomography*, the problem of inferring network-level anomalies from widely available data aggregates. Our major advances are:

1. We introduced a powerful framework for anomography that cleanly separates the anomaly detection component from the inference component. The framework opens up a wide field for innovation and for the development of families of new algorithms. The novel method of Lakhina *et al.* based on PCA falls within the framework.

2. Within the framework, we put forward a number of novel algorithms, taking advantage of the range of choices for anomaly detection and inference components and choosing between temporal versus spatial approaches.

3. We developed a new *dynamic anomography* algorithm, which tracks both routing and traffic measurements, and so enables alerting with high fidelity on traffic matrix anomalies, without alerting on internal routing changes that leave the traffic matrix relatively stable. As routing changes are often due to normal internal self-healing behavior separating these changes from intrinsic traffic anomalies is advantageous. An additional benefit of dynamic anomography is that is robust to missing data, an important operational reality.

4. Using extensive data from Internet2's Abilene network and a Tier-1 ISP, we evaluated these anomography methods. The findings are encouraging. Specifically, the results indicate that the new set of *temporal* anomography methods introduced here have better fidelity, particularly when using l^1 minimization for the inference step. Dynamic anomography using ARIMA based methods and l^1 norm minimization shows uniformly high fidelity (low false positive and false negatives) and high robustness (to routing changes and missing or corrupted data).

While we believe our work represents a significant advance in the state of the art, we recognize that the the ultimate test of performance is significant operational experience: utility is bringing to light in the field new anomalies that were "flying under the radar" of other techniques, while producing very few false alarms. Our larger goal in future work is to explore the feasibility and performance of automated traffic management systems, which incorporate anomaly detection, root cause diagnosis and traffic and route control for operational networks.

Notes

[1]Note that the link load vector $\mathbf{b}_j$ includes the aggregated traffic at different ingress/egress points; the corresponding rows in A_j encode the OD flows that enter/exit the network at these points.

[2]We have verified that Pseudoinverse and Sparsity-Greedy work even worse than Sparsity-L1 for PCA.

References

[1] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *ACM Internet Measurement Workshop*, Nov. 2002.

[2] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting and Control*. Prentice-Hall, Englewood Cliffs, 1994.

[3] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Varlang, 2nd edition, 2002.

[4] J. D. Brutag. Aberrant behavior detection and control in time series for network monitoring. In *14th Systems Administration Conference (LISA 2000)*, Dec. 2000.

[5] J. Cao, D. Davis, S. V. Wiel, and B. Yu. Time-varying network tomography. *J. Amer. Statist. Assoc*, 95(452):1063–1075, 2000.

[6] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41:909–996, 1988.

[7] I. Daubechies. *Ten Lectures on Wavelets*, volume 41 of CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1992.

[8] D. L. Donoho. For most large underdetermined systems of equations, the minimal l1-norm near-solution approximates the sparsest near-solution, Aug. 2004. `http://www-stat.stanford.edu/~donoho/Reports/`.

[9] D. L. Donoho. For most large underdetermined systems of equations, the minimal l1-norm solution is also the sparsest solution, Sept. 2004. `http://www-stat.stanford.edu/~donoho/Reports/`.

[10] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: Traffic engineering for IP networks. *IEEE Network Magazine*, pages 11–19, Mar./Apr. 2000.

[11] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, 9(3):265–279, 2001.

[12] A. Graps. Amara's wavelet page, 2004. `http://www.amara.com/current/wavelet.html`.

[13] M. Grossglauser, N. Koudas, Y. Park, and A. Variot. FALCON: Fault management via alarm warehousing and mining. In *NRDM 2001 Workshop*, May 2001.

[14] A. Gunnar, M. Johansson, and T. Telkamp. Traffic matrix estimation on a large IP backbone: A comparison on real data. In *ACM Internet Measurement Conference*, Oct. 2004.

[15] C. Hood and C. Ji. Proactive network fault detection. *IEEE Trans. Reliability*, 46(3):333–341, 1997.

[16] Katzela and Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 3(6):753–764, 1995.

[17] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *ACM Internet Measurement Conference*, Oct. 2003.

[18] A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *ACM Internet Measurement Conference*, Oct. 2004.

[19] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM*, Aug. 2004.

[20] A. Lakhina, K. Papagiannaki, C. D. Mark Crovella, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *ACM SIGMETRICS*, 2004.

[21] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, 2nd edition, 2001.

[22] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *ACM SIGCOMM*, Aug. 2002.

[23] A. Nucci, R. Cruz, N. Taft, and C. Diot. Design of IGP link weights for estimation of traffic matrices. In *IEEE Infocom*, Mar. 2004.

[24] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *27th Annual Asilomar Conference on Signals, Systems, and Computers*, 1993.

[25] M. Roughan, T. Griffin, M. Mao, A. Greenberg, and B. Freeman. IP forwarding anomalies and improving their detection using multiple data sources. In *ACM SIGCOMM Workshop on Network Troubleshooting*, pages 307–312, Sept. 2004.

[26] SAS 9 online document. Equations for the smoothing models, Jan. 2004. `http://support.sas.com/91doc/getDoc/hpfug.hlp/hpfdet_sect7.htm`.

[27] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of OSPF behavior in a large enterprise network. In *ACM Internet Measurement Workshop*, 2002.

[28] C. Tebaldi and M. West. Bayesian inference on network traffic using link count data. *J. Amer. Statist. Assoc*, 93(442):557–576, 1998.

[29] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic matrix reloaded: Impact of routing changes renata teixeira. In *Workshop on Passive and Active Measurements (PAM)*, 2005.

[30] M. Thottan and C. Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network*, Sept/Oct 1998.

[31] Y. Vardi. Network tomography: estimating source-destination traffic intensities from link data. *J. Amer. Statist. Assoc.*, 91(433):365–377, 1996.

[32] A. Ward, P. Glynn, and K. Richardson. Internet service performance failure detection. *ACM SIGMETRICS Performance Evaluation Review archive*, 26(3):38–43, Dec. 1998.

[33] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan. Network anomography. UT-Austin Technical Report, 2005. `http://www.cs.utexas.edu/~yzhang/papers/`.

[34] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *ACM SIGMETRICS*, June 2003.

[35] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *ACM SIGCOMM*, Aug. 2003.

Combining Filtering and Statistical Methods for Anomaly Detection

Augustin Soule
LIP6-UPMC

Kavé Salamatian
LIP6-UPMC

Nina Taft
Intel Research

Abstract

In this work we develop an approach for anomaly detection for large scale networks such as that of an enterprize or an ISP. The traffic patterns we focus on for analysis are that of a network-wide view of the traffic state, called the traffic matrix. In the first step a Kalman filter is used to filter out the “normal” traffic. This is done by comparing our future predictions of the traffic matrix state to an inference of the actual traffic matrix that is made using more recent measurement data than those used for prediction. In the second step the residual filtered process is then examined for anomalies. We explain here how any anomaly detection method can be viewed as a problem in statistical hypothesis testing. We study and compare four different methods for analyzing residuals, two of which are new. These methods focus on different aspects of the traffic pattern change. One focuses on instantaneous behavior, another focuses on changes in the mean of the residual process, a third on changes in the variance behavior, and a fourth examines variance changes over multiple timescales. We evaluate and compare all of these methods using ROC curves that illustrate the full tradeoff between false positives and false negatives for the complete spectrum of decision thresholds.

1 Introduction

Traffic anomalies such as attacks, flash crowds, large file transfers and outages occur fairly frequently in the Internet today. Large enterprise networks often have a security operations center where operators continuously monitor the network traffic hoping to detect, identify and treat anomalies. In smaller networks, these tasks are carried out by general network administrators who are also carry out other day-to-day network maintenance and planning activities. Despite the recent growth in monitoring technology and in intrusion detection systems, correctly detecting anomalies in a timely fashion remains a challenging task.

One of the reasons for this is that many of today’s security solutions yield equipment that collects and analyzes traffic from one link at a time. Similarly many research efforts consider anomaly detection on a per link basis [2, 8, 3]. To detect traffic anomalies one typically seeks to characterize, or build a model, of what constitutes normal behavior. After filtering out normal looking traffic, anomaly detection methods analyze the residual traffic pattern for deviations. Considering only one link is limiting. Since any flow will traverse multiple links along its path, it is intuitive that a flow carrying an anomaly will appear in multiple links, thus increasing the evidence to detect it. Instead in this paper, we focus on using data from all the links in an enterprise or ISP network simultaneously. Since any anomaly has to traverse multiple links on route to its destination, an anomaly has the potential to be visible in any of the links its traverses. Since we cannot know in advance where anomalies will originate, nor the path they will take, it is advantageous to consider the behavior of all the links in an enterprise simultaneously when developing both a model of ”normal” traffic and a method for analyzing the ”residuals”.

A traffic matrix is a representation of the network-wide traffic demands. Each traffic matrix entry describes the average volume of traffic, in a given time interval, that originates at a given source node and is headed towards a particular destination node. In an enterprise network these nodes may be computers, whereas in an ISP network the end nodes can be routers. In this paper we propose to use predictions of traffic matrix behavior for the purposes of anomaly detection.

Since a traffic matrix is a representation of traffic volume, the types of anomalies we might be able to detect via analysis of the traffic matrix are volume anomalies [12]. Examples of events that create volume anomalies are denial-of-service attacks (DOS), flash crowds and alpha events (e.g., non-malicious large file transfers), as well as outages (e.g., coming from equipment failures).

Obtaining traffic matrices was originally viewed as a challenging task since it is believed that directly measuring them is extremely costly as it requirements the deployment of monitoring infrastructure everywhere, the collection of fine granularity data at the flow level, and then the processing of large amounts of data. However in the last few years many inference based techniques have been developed (such as [22, 23, 19, 18, 20, 6] and many others) that can estimate traffic matrices reasonably well given only per-link data such as SNMP data (that is widely available). These techniques focus on estimation and not prediction.

In this paper we build upon one of our previous techniques [20, 18] for traffic matrix estimation by using it to provide predictions of future values of the traffic matrix. A traffic matrix is a dynamic entity that continually evolves over time, thus estimates of a traffic matrix are usually provided for each time interval (e.g., most previous techniques focus on 5 or 10 minute intervals). We predict the traffic matrix one step (e.g., 5 minutes) into the future. One of the key ideas behind our approach lies in the following observation. Five minutes after the prediction is made, we obtain new link-level SNMP measurements, and then estimate what the actual traffic matrix should be. We then examine the difference between our prediction (made without the most recent link-level measurements) and the estimation (made using the most recent measurements). If our estimates and predictor are usually good, then this difference should be close to zero. When the difference is sizeable we become suspicious and analyze this residual further to determine whether or not an anomaly alert should be generated.

We compare four different methods for signalling alerts when analyzing residual traffic. The simplest method compares the instantaneous residual traffic to a threshold. The second method considered is a small variation on the deviation score idea presented in [2]. Their key idea is to compare a local (temporally) variance calculation with a global variance assessment. The deviation score used in [2] is computed using output signals of a wavelet transform applied to IP flow level data from a single link. We apply this idea of comparing the local to the global variance on our filtered residual signal. In our third scheme, we apply wavelet analysis only on the filtered traffic (in [2] wavelet analysis is applied directly on the original signal). We signal an alert when the detail signal (now a type of residual) at each of a few different timescales exceeds a threshold. We raise an alarm only if the threshold is exceeded at multiple timescales. The fourth method uses a generalized likelihood ratio test to identify the moment an anomaly starts, by identifying a change in mean rate of the residual signal. These last two methods, introduced here for the first time, are particular applications of known statistical techniques to the anomaly detection domain.

Our approach is different from other approaches in that usually anomaly detection is performed directly on monitored data that is captured at the target granularity level. Instead we perform anomaly detection on origin-destination (OD) flows, a granularity of data that we infer from other measurements (link statistics). Our study shows that it is possible to follow such an approach towards a positive outcome.

To validate our methods we use both real data from the Abilene network and a synthetic anomaly generator that we developed. These two approaches are complementary as their advantages and disadvantages are opposite. The advantage of evaluating using real world traces is that we test our methods on actual anomalies that have occurred in the Internet. The disadvantage of using only collected traces is that the statistical parameters of the anomaly cannot be varied. One cannot therefore identify the limits of a method. For example, one cannot ask ”would we still detect the anomaly if its volume were lower?”. Using synthetically generated anomalies in which we carefully control the anomaly parameters, we can stress test and identify the limits of an algorithm. However the synthetic anomalies are limited because we have no evidence of them in the Internet. Our approach to validation thus employs both of these approaches in order to extract the benefits of each.

We use ROC (Receiver Operating Characteristic) curves as our key evaluation criteria. ROC curves have received wide usage in medical diagnosis and signal detection theory, but relatively little in network security. A ROC curve is a graphical representation of the tradeoff between the false positive and false negative rates for every possible decision threshold. We include a brief description of the meaning and theory behind ROC curves to illustrate a general methodology for analysing network security solutions. We use these to compare our four solutions. The advantage of this approach is that it permits scheme comparison throughout the entire range of decision thresholds. This eliminates the difficulty that arises when one tries to compare methods each of which uses a particular and seemingly ad hoc threshold choice. In addition, we also present the performance of these methods in terms of their detection time. This is important as most anomaly detection methods incur some lag time before reaching a decision. Finally we assess the false positive and false negative rates our schemes yield as the volume of an anomaly is varied from low-volume anomalies to high-volume ones.

The most important, and only, work to date that uses a network-wide perspective for volume anomaly detection was that of [12]. In this work, the authors used the ensemble of all the links in a network and performed Principle Components Analysis to reduce the dimensionality of the data. They illustrate that by projecting onto a small number of principal components one could filter out the ”normal” traffic. The traffic projected onto the remaining components is analyzed for anomalies using a G-statistic test on the predictive error. While our paper essentially tackles the same problem, our work differs in numerous ways: i) we process the incoming link data using kalman filters rather than PCA analysis and generate traffic matrix predictions; ii) the granularity we focus on is that of OD flows whereas they use link data when analyzing residuals. (Note, they use the OD flows as a secondary step, after detecting an anomaly, in order to identify the source); iii) they consider a single test on the residual traffic whereas we propose two new ones and conduct a comparative evaluation of four schemes; iv) our method for validation differs since we supplement the Abilene data with synthetic anomaly testing; v) our evaluation is different because we make use of ROC curves for evaluation, examine detection lag times as well as sensitivity to anomaly volume sizes.

Section 2 describes how we model the OD flows and our solution for traffic matrix prediction. The methods for analyzing filtered traffic and determining how to detect an anomaly are presented in Section 3. We discuss our approach to validation and fully describe our synthetic anomaly generator in Section 4. All of our evaluations and the results are shown in Section 5.

2 Modeling Normal Traffic

We assume that the monitoring infrastructure in our network can easily obtain per-link statistics on byte counts (as in SNMP today). From this we want to infer the traffic matrix that includes all pairs of origin-destination (OD) flows. This is the classic traffic matrix estimation problem. If we design a realistic model for the evolution of the network's traffic matrix, then we can use this to filter our usual behavior. For the sake of completeness we now summarize our linear dynamic state space model for the OD flows and our Kalman filter method for estimating the traffic matrix. This was originally presented in [18]. We expand on our previous work by illustrating how this can be used to make future predictions of the traffic matrix and describe the resulting residual processes that can be obtained when filtering via this approach.

Since the OD flows are not directly observable (measurable with today's technology) from the network, we refer to them as *hidden network states* or simply as *network states*. The link load levels (e.g., total bytes per unit time) are directly observable in networks, and are captured via the SNMP protocol that is widely deployed in most commercial networks today. Because the total traffic on a link is the sum of all the OD flows traversing that link, the relationship between SNMP data and OD flows can be expressed by the linear equation $Y_t = A_t X_t + V_t$, where Y_t represents the vector of link counts vector at time t, and X_t is the OD flows organized as a vector (hidden network states). A_t denotes the routing matrix whose elements $a_t(i,j)$ are 1 if OD flow j traverses link i, and zero otherwise. (In some networks fractional routing is supported.) The term V_t captures the stochastic measurement errors associated with the data collection step. All these parameters are defined for a general discrete time t.

To capture the dynamic evolution of OD flows we need a model that specifies X_{t+1} as a function of X_t. We seek a model that can be used for prediction of the OD flows one step into the future. Providing an efficient model that captures traffic dynamics is not so simple. It has been observed that traffic entering the network is characterized by highly variable behavior in time [13]. There are many sources of this variability, including daily periodic behavior, random fluctuations with relatively small amplitude, and occasional bursts. Sudden changes in the traffic are not uncommon and can be related to different benign causes such as the addition of new customers, network equipment failures, flash crowds or to malicious activities such as attacks conducted against the network. Ignoring the attacks for the moment, our model for OD flows must be rich enough to incorporate these sources of variability for normal traffic. It is also known that both temporal correlations within a single OD flow exist, and that spatial correlations across some OD flows occurs [18].

We adopt a linear state space model to capture the evolution of OD flows in time. This predictive model relates the network state X_{t+1} to X_t as follows: $X_{t+1} = C_t X_t + W_t$, where the state transition matrix C_t captures temporal and spatial correlations in the system, and W_t is a noise process that accounts for both the randomness in the fluctuation of a flow, and the imperfection of the prediction model. Linear stochastic predictive models, combined with Gaussian noise, have been successfully applied to a large spectrum of monitoring problems.

The matrix C_t is an important element of the system. A diagonal structure for C_t indicates that only temporal correlations are included in the model of an OD flow. When C_t has off-diagonal elements that are non-zero, then spatial correlation across OD flows have been incorporated into the model. For traffic matrix estimation, using a non-diagonal matrix for C_t is preferable so that one can benefit from incorporating spatial correlation (as used in [20]). When traffic matrix estimation is carried out, the main task is that of taking a total byte count for each link and partitioning it among the the multiple OD flows traversing that link. When an anomaly occurs on a link, it is possible for an anomaly (originating within one OD flow) to get spread across all the OD flows on that link during the estimation procedure. To avoid this phenomenon, that would make it more difficult to detect anomalies in OD flows, we use a diagonal structure for C_t (unlike the model used in [18]).

Putting the above elements together, our complete model is that of a linear state space dynamical system, that relates the observables (Y_t) to the unobservables (X_t), and is given by,

$$\begin{cases} X_{t+1} &= C_t X_t + W_t \\ Y_t &= A_t X_t + V_t \end{cases} \tag{1}$$

We assume both the state-noise W_t and the measurement-noise V_t to be uncorrelated, zero-mean Gaussian white-noise processes and with covariance matrices Q_t and R_t:

$$E[W_k W_l^T] = \begin{cases} Q_k, & \text{if } k = l \\ 0, & \text{otherwise} \end{cases}$$

$$E[V_k V_l^T] = \begin{cases} R_k, & \text{if } k = l \\ 0, & \text{otherwise} \end{cases}$$

$$E[W_k V_l^T] = 0 \quad \forall k, l \tag{2}$$

These assumptions might appear restrictive however a large body of research in the control theory literature has been devoted to Kalman filtering robustness. The lessons learned from this literature are that because of the feedback mechanism, and ongoing readjustment of estimated values, Kalman Filters are robust to model imprecision as well as to some deviation from gaussianity in the noise. The rule of thumb for reaching a certain level of robustness is to use noise with slightly larger variance for W_t than obtained by direct evaluation of noise.

Given the above assumptions and a set of observations $\{Y_1, ..., Y_{t+1}\}$, the task is to determine the estimation filter that at the $(t+1)$-*st* instance in time generates an optimal estimate of the state X_{t+1}, which we denote by $\hat{X}_{t+1}$. Optimality is defined in the sense of Minimum Variance Error Estimator that is defined as follows:

$$E[||X_{t+1} - \hat{X}_{t+1}||^2] = E[(X_{t+1} - \hat{X}_{t+1})^T (X_{t+1} - \hat{X}_{t+1})] \tag{3}$$

The classical tool for dealing with this type of problem is the well known Kalman Filter [10]. It addresses the general problem of trying to estimate a discrete state vector when the observations are only a linear combination of this underlying state vector. The Kalman filter estimates the system state process by using a two step approach, that iterates for each time t. We use $\hat{X}_{t|i}$ we refer to the estimation of X_t based on time $i, t \geq i$. (We introduce here the more general case of time-varying systems, where all the parameters are indexed by time.)

• **Prediction Step**: Let $\hat{X}_{t|t}$ denote the estimate of the state at time t given all the observations up to time t (i.e. Y^t). This term has a variance that is denoted by $P_{t|t}$. Let $\hat{X}_{t+1|t}$ denote the one step predictor. This prediction is made using all the observed data up to time t. Since the model $X_{t+1} = C_t X_t + W_t$ includes the noise term W_t (with covariance Q_t), this prediction will have some associated variability, that is denote as $P_{t+1|t}$. In the prediction step, we are given $\hat{X}_{t|t}$ and $P_{t|t}$, and compute both our prediction, and the variance of this prediction, as follows.

$$\begin{cases} \hat{X}_{t+1|t} & = C_t \hat{X}_{t|t} \\ P_{t+1|t} & = C_t P_{t|t} C_t^T + Q_t \end{cases} \tag{4}$$

• **Estimation Step**: In this step, the kalman filter updates the state estimate $X_{t+1|t+1}$, and its variance ($P_{t+1|t+1}$) by using a combination of their predicted values and the new observation Y_{t+1}. The new estimate at time $t+1$ is given by,

$$\begin{cases} \hat{X}_{t+1|t+1} = & \hat{X}_{t+1|t} + K_{t+1}[Y_{t+1} - A_{t+1}\hat{X}_{t+1|t}] \\ P_{t+1|t+1} = & (I - K_{t+1}A_{t+1})P_{t+1|t}(I - K_{t+1}A_{t+1})^T \\ & +K_{t+1}R_{t+1}K_{t+1}^T \end{cases} \tag{5}$$

The new estimate at time $t+1$ for $\hat{X}_{t+1|t+1}$ is computed using the prediction from the previous time instant $\hat{X}_{t+1|t}$ that is adjusted by a correction factor. Consider the latter part of this equation. By multiplying our prediction $\hat{X}_{t+1|t}$ by A_t, we generate a prediction for the link counts $\hat{Y}_{t+1}$. Hence the term in brackets $[Y_{t+1} - A_{t+1}\hat{X}_{t+1|t}] = Y_{t+1} - \hat{Y}_{t+1}$ is the error in our prediction of the link counts. This term is multiplied by the matrix K_{t+1} that is called Kalman gain matrix. It is obtained by minimizing the conditional mean-squared estimation error $E[\tilde{X}_{t+1|t+1}^T \tilde{X}_{t+1|t+1} | Y^t]$ where the estimation error is given by $\tilde{X}_{t|t} = \hat{X}_{t|t} - X_t$. By applying some basic linear algebra, we can write it as:

$$K_{t+1} = P_{t+1|t} A_{t+1}^T [A_t P_{t+1|t} A_{t+1}^T + R_{t+1}]^{-1} \tag{6}$$

Hence this second step takes the new observation of Y when it becomes available, and corrects its previous prediction. The above equations together with the initial conditions of the state of the system $\hat{X}_{0|0} = E[X_0]$ and the associated error covariance matrix $P_{0|0} = E[(\hat{X}_{0|0} - X_0)(\hat{X}_{0|0} - X_0)^T]$ define the discrete-time sequential recursive algorithm, for determining the linear minimum variance estimate, known as Kalman Filter.

In our previous paper [18], the traffic matrix is populated (i.e. estimated) using $\hat{X}_{k+1|k+1}$. Nevertheless, it is clear that the Kalman filter gives more information than only estimates. Using the predictive ability of the filter it is possible to estimate the future evolution of the traffic matrix. The correction step in Equation (5) essentially captures the part of the process that our model could not predict. It is this unpredictable part that we want to track for anomaly detection. Based on the study in [18], we know that the Kalman filter method for estimating the traffic matrix works well. Hence most of the time, the correction factors are negligible. Now if at some time instant we see a large correction of our prediction , we could flag this as anomalous and generate an alert.

We are thus motivated to examine the errors that our one-step predictor generates. The errors in our prediction of the link values are denoted by,

$$\epsilon_{t+1} = Y_{t+1} - A_{t+1}\hat{X}_{t+1|t},$$

In Kalman filtering terminology this error is typically the innovation process. It is the difference between the observed (measured) value Y_{t+1} and its prediction $A_{t+1}\hat{X}_{t+1|t}$. The innovation process ϵ_t is considered to be white gaussian noise with a covariance matrix given by :

$$E\left[\epsilon_{t+1}\epsilon_{t+1}^T\right] = A_{t+1}P_{t+1|t}A_{t+1}^T + R_{t+1}. \qquad (7)$$

Since in our case, we are interested in anomalies in the OD flows, we can define, by extension, the **residual** η_{t+1},

$$\eta_{t+1} = \hat{X}_{t+1|t+1} - \hat{X}_{t+1|t} = K_{t+1}\epsilon_{t+1},$$

that is the difference between the new estimate of the state ($\hat{X}_{t+1|t+1}$), corrected using the most recent measurement at time $(t+1)$, and its prediction $\hat{X}_{t+1|t}$ made based only on information available up to time t. It is also a measure of the new information provided by adding another measurement in the estimation process. Using Equation (5), we can see that the error in the OD flow estimate is related to the error in the link estimate via $\eta_{t+1} = K_{t+1}\epsilon_{t+1}$.

This is also a zero-mean gaussian process, whose variance S_{t+1} can be easily derived as

$$S_{t+1} = E[\eta_{t+1}\,\eta_{t+1}^T] = K_{t+1}(A_{t+1}P_{t+1|t}A_{t+1}^T + R_{t+1})K_{t+1}^T \qquad (8)$$

The residual process can be shown to be asymptotically uncorrelated, *i.e.* $E\left[\eta_t\,\eta_l^T\right] = 0,\ t \neq l$. This can be understood by observing that asymptotically the gain matrix of Kalman filter K_{t+1} converge to a fixed point $\bar{K}$. The residual is an important measure of how well an estimator is performing. A non-zero residual could mean that an anomaly has occurred, and in the next section 3, we present a few schemes for further examining this residual time series to detect anomalies.

In this section, we presented the Kalman filtering method in its general settings under non-stationary assumptions. In the following sections, we will assume a stationary situation where the matrices A, C, Q and R are constant in time, making it possible to drop their subscripts. However, the rest of the methodology presented in this paper can easily be generalized to incorporate time dependency.

There is an issue of calibration for using such a Kalman filter model because the matrices C, Q and R need to be calibrated. We developed and presented in [20] an Expectation Maximization based approach for calibrating these matrices. In [20] we showed that for reliable OD flow estimation we need to recalibrate the Kalman filter every few days when the underlying model changes. When there are anomalies, this might suggest that the model should be recalibrated every time an anomaly occurs. However, one interesting result of this current paper is that this recalibration step is often not needed if the goal is just anomaly detection. For example, in applying our anomaly detection schemes on the Abilene data, we found that no recalibration was needed for 7 days (covering 74 anomalies). Hence the requirements for recalibration appear to be stronger for traffic matrix estimation than for anomaly detection.

3 Analyzing Residuals

Before explaining our four methods for examining residuals to look for anomalies, we discuss some important issues regarding sources of errors, understanding the meaning of decision thresholds, and how they are selected. In doing so, we explain our methodology for comparing different anomaly detection schemes.

There are two sources of errors that can appear in the residual process. One is from errors in the underlying traffic model, while the second will come from anomalies in the traffic. Suppose, for a moment, that we consider any general random process Z_t that we want to check for anomalies. Let $\hat{Z}_t$ denote our prediction for this process based upon a model. Since our model may not be exact, we let ζ_t denote the expected prediction error, a zero-mean random variable with known covariance. If we define ξ_t as the anomaly term at time t, we can write :

$$Z_t = \hat{Z}_t + \zeta_t + \xi_t.$$

In this equation ξ_t is a random variable accounting for the unexpected change caused by the anomalies, *i.e.* $\xi_t = 0$ if there are no anomalies and $\xi_t \neq 0$ when there is an anomaly.

There is an important decision to be made as to which data granularity to examine in order to try to observe anomalies. We can consider either looking at the prediction errors observed on the link data Y_t or the estimation errors on the OD flows X_t. Our experience showed us that detection schemes work better when operating at the granularity level of the OD flow rather than at that of the link. Although we cannot observe the OD flow directly, we can observe the error in our prediction of the OD flow and that turns out to be plenty sufficient for our purposes. We point out that the four schemes we discuss for examining errors can be applied to either type of error. These methodologies require only that we understand the covariance process of the associated ζ_t process.

To detect anomalies on the SNMP link counts, one should use the statistics of the innovation process in place of the statistics of ζ. This is readily available in our model since it is equivalent to the statistics of the innovation process in the Kalman Filter. The innovation obtained as the output of the Kalman filter is exactly the prediction error $\zeta_t + \xi_t$.

Anomaly detection on OD flows is more tricky as the prediction error is not directly observable (as OD flows are hidden). However, the good news is that the covariance of ζ is known and equal to $P_{t+1|t+1}$. Moreover, the residual $\eta_{t+1} = \hat{X}_{t+1|t+1} - \hat{X}_{t+1|t} = K_{t+1}\epsilon_{t+1}$ can be observed and its covariance can be derived as S_{t+1}. And last but not least the estimation error ζ and the residual η are correlated gaussian processes, *i.e* one might use one for estimating the other and the least squared error estimator is :

$$\zeta_t + \xi_t \approx -K_t A_t P_{t|t-1} S_t^{-1} \eta_t \tag{9}$$

The approximation comes from the fact that this is just an estimation of an unobserved value (the OD flows estimation error) based on an observed value (residual).

3.1 Anomaly detection as a statistical test

We now wish to illustrate how any anomaly detection scheme can be viewed as a statistical hypothesis test. To do this, we first explain how such tests are evaluated. The tested are evaluated by exploring the fundamental tradeoff between the false positive and false negative rates. Hypothesis testing explains how to pick decision thresholds when faced with balancing this particular tradeoff.

All four of the schemes we use to evaluate the residuals rely on the selection of a threshold that is used to decide whether or not an alarm is raised. In fact, any anomaly or change detection method will require that a threshold be selected. In our evaluation of these methods we consider all possible thresholds for each method. We do this by assessing the performance of our method using Receiver Operation Characteristic (ROC) curves.

ROC curves have been developed in the context of signal detection [5], and have been widely used for medical analysis purposes [24]. ROC curves are useful because the describe the full tradeoff between false positives and false negatives over the complete spectrum of operating conditions (i.e., decision threshold settings). In an ROC curve, we plot the false positive rate on the x-axis and one minus the false negative rate on the y-axis. The y-axis thus represents the true positives (the anomalies we want to catch). An algorithm is considered very good if its ROC curve climbs rapidly towards the upper left corner of the graph. This means that we detect a very high fraction of the true anomalies with only a few false positives. Sample ROC curves can be seen in Figure 1 (to be fully explained later).

To quantify how quickly the ROC curve rises to the upper left hand corner, one simply measures the area under the curve. The larger the area, the better the algorithm. ROC curves are essentially parametric plots as each point on the curve corresponds to a different threshold. Each point illustrates a particular tradeoff between false positives and false negatives. Each algorithm results in one curve, and by comparing these curves we can compare algorithms. The curve with the largest area underneath it corresponds to the better algorithm. Since each curve represents the entire range of thresholds, we can compare algorithms throughout their entire region.

ROC curves are grounded in statistical hypothesis testing. As mentioned earlier, any anomaly detection method will at some point use a statistical test to verify whether or not a hypothesis (e.g., there was an anomaly) is true or false. Recall that ξ_t is our residual process and should be zero (or roughly zero) when there is no anomaly. We can form the hypothesis $\mathcal{H}_0 : \xi_t = 0$ for the case when there is no anomaly. We can form an alternate hypothesis $\mathcal{H}_1 : \xi_t! = 0$ for the case when there an anomaly occurs. This last hypothesis is difficult to handle mathematically, so for the sake of simplicity of exposition, we rewrite the alternate hypothesis as $\mathcal{H}_1 : \xi_t = \mu$. (Conceptually we can continue to think of this as the case when an anomaly occurs). The random variable ξ_t in each hypothesis is assumed to have some distribution. Upon observing a sample of this random variable we compare it to a threshold to decide if we reject $\mathcal{H}_0$ (thereby accepting $\mathcal{H}_1$) or vice versa.

Let FPR denote the false positive rate, the probability that we detect an anomaly given there was no anomaly. Put otherwise, this is the likelihood that we reject $\mathcal{H}_0$ when it was true. The false negative rate, FNR, is the probability that we detect nothing when an anomaly occurs (or the likelihood that we accept $\mathcal{H}_0$ when we should have rejected it). In order to decide whether or not to accept $\mathcal{H}_0$, we compare our observation of ξ_t to a threshold. The Neyman-Pearson criteria says that we should construct this decision threshold to maximize the probability of detection (true positives) while not allowing the probability of false alarm to exceed some value α.

The optimization problem to solve is to find the maximum probability of detection (1-FNR) such that $FPR \leq \alpha$. The likelihood ratio is defined as the ratio of FPR/FNR. The Neyman-Pearson lemma says that the optimal decision threshold is one that satisfies the likelihood ratio test.

$$\frac{FPR}{FNR} \leq T(\alpha)$$

In solving for $T(\alpha)$ (i.e., deriving the curve), each point of this curve corresponds to one value of the decision threshold. In practice, this curve is plotted as the correct detection rate,*i.e.* $1 - FNR$ as a function of false positive rate FPR thus yielding the ROC curve.

For a fixed $FPR = \alpha$, all values $1 - FNR \leq \beta^*$ are achievable by a non-optimal anomaly detector, or equivalently all points below the optimal ROC curve can be achieved. The ROC curve can be derived analytically typically only under simple assumptions (such as ξ_t is Gaussian). In this case the derived curve is an optimal curve. The optimal curve is not a perfect solution (i.e., 100% true positive detection and 0% false positives) because usually there is some inherent noise in the process this limits the best decision one can make.

As a simple example, consider the case when ξ_t is a gaussian random variable with a cumulative distribution given by Φ. The ROC curve for the hypothesis $\mathcal{H}_0 : \xi_t = 0$ *vs.* $\mathcal{H}_1 : \xi_t = \mu$ is given by :

$$1 - FNR = 1 - \Phi(\Phi^{-1}(FPR) - \mu)$$

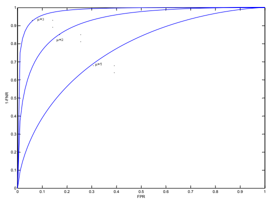


Figure 1: Optimal ROC curve for a gaussian hypothesis testing between $\mathcal{H}_0 : \zeta = 0$ *vs.* $\mathcal{H}_1 : \zeta = \mu$.

Fig. 1 shows the ROC curve for this case for three different alternate hypotheses $\mathcal{H}_1$. In practice, the optimal ROC curves cannot be derived, which limits our ability to see how far a particular detection scheme is from optimal, where optimal is determined based on the underlying noise of the system. However since each scheme yields a different ROC curve, these remain a powerful means of comparison across schemes. If one curve has less area beneath it, then it is clearly inferior, regardless of the threshold level selected.

3.2 Basic analysis using Variance

The first anomaly detector that will be described is also the simplest one. As seen previously in normal operational condition one might assume that $\xi_t = 0$ and that the prediction error ζ_t follows a process with mean 0 and known variance. Under the situation that the statistics of ζ_t, the prediction error are fully known, it is easy to construct a statistical test following the Neyman-Pearson theorem. For this purpose we might use the construction given in Eq. 9.

The approach consists of constructing the process $\tau_t = -K_t A_t P_{t|t-1} S_t^{-1} \eta_t$ and rising an alarm for an OD pair i whenever $\tau_{ti} > T \times \sqrt{(P_{t+1|t+1})_{ii}}$ where T is the threshold. Actually, this approach verifies if the prediction error is inside a confidence interval. This anomaly detector is the optimal one for the case where $\zeta_t + \eta_t$ follow a gaussian distribution. However, if this hypothesis in not precisely true (as frequently in practice), application of this anomaly detector will lead to a ROC curve that is lower than the optimal one.

An interesting property of this method is that the test is verified as soon as a new observation has been processed by the Kalman filter and it can therefore trigger an anomaly very fast. However the drawback of the approach is that each test is being done independently of past observations. This might lead to high false positive rate when the process ζ_t has a heavier tail than the gaussian. One might want to have a less sensitive approach that will not raise an alarm based on only one observation diverging from the bound.

3.3 CUSUM and Generalized Likelihood Ratio test

The previous method missed an essential fact, since we are in the context of random processes, tests executed at each time t are not independent. The classical approach for detecting a change in a random process is the CUSUM (Cumulative Summation) method and its variants [4]. The main intuition behind the CUSUM method is that when a change occurs the log-likelihood ratio of an observation y_i, defined as $s_i = \log \frac{L_1(y)}{L_0(y)}$, shifts from a negative value to a positive one (as after the change hypothesis $\mathcal{H}_1$ becomes more likely). This means that the log-likelihood of observing a sequence of N observations $\{y_0^{N-1}\}$, defined as $S_{N-1} = \sum_{i=0}^{N-1} s_i$, that was decreasing with N, begins to increase after the change. The minimum value of S_j gives an estimate of the change point. Therefore a simple statistical test for change detection consists of testing whether :

$$S_k - \min_{0 \le j \le k} S_j > T,$$

where S_k is the log-likelihood ratio defined previously and T is a threshold. After a change has been detected, the time of change can be estimated as :

$$\hat{t}_c = \arg \min_{0 \le j \le k} \{S_j\}$$

The previously described CUSUM algorithm has been widely used for anomaly detection. However it suffers from a key drawback. It is stated in the context of a simple hypothesis, where the alternative hypothesis $\mathcal{H}_1$ should be completely defined, *i.e.* the level of the change or in other terms the intensity of the anomaly should be known *a priori*. However in practical settings, this is exactly unknown as by definition anomalies are not predictable.

A solution for this issue is provided by the *General Likelihood Ratio Test*. In this approach the level of change in the CUSUM algorithm is replaced by its maximum likelihood estimate. To describe the approach let's fix a scenario. Suppose an anomaly occurs and this results in a shift in the mean of the residual process. After the shift, the estimation error will no longer be a zero mean random variable of variance σ (σ is assumed to be known), but instead is translated to a mean μ, that is unknown, and the same variance. The GLR algorithm uses a window of estimation error $\{\tau_j^{j+N-1}\}$ and applies for each $i, j \le i \le j+N-1$ the following test. It first estimates the mean of the estimation error over the window $\{i, \dots j+N-1\}$ as

$$\hat{\mu} = \frac{1}{j+N-1-i} \sum_{l=i}^{j+N-1} \tau_l$$

It then performs a simple CUSUM test with $\hat{\mu}$ as the level change value and we raise an alarm if a change is detected. We implemented here a variant of the classical GLR method described in [7]. This method is very powerful since there exists a proof that this is the best estimator when level change μ and variance σ are unknown. However its main drawback is that it adds some delay for the detection of the anomaly since it needs some observations after the anomaly to estimate the deviation level. The detection delay will not be constant and will depend on the anomaly. For example, the effect of small volume anomalies on the mean will propagate slowly and thus may not be detected as quickly as large volume anomalies.

3.4 Multiscale analysis using variance

Multi scale analysis has been proposed as a promising approach to make robust anomaly detectors and is now commonly accepted as a powerful tool. The rational behind using multiscale analysis is that anomalies should appear at different time scales and by monitoring these multiple scales one should be able to reduce the False Positive Rate, because a change appearing on only one time scale will not trigger an alarm.

We implemented a multi-scale analysis based on a cascade decomposition of the original signal τ_t into a low frequency approximation a_t^L and a cascade of details d_t^i. The multi-scale decomposition lead to the following relation :

$$\tau_t = a_t^L + \sum_{i=1}^{L} d_t^i.$$

where :

$$\begin{aligned} d_t^i &= \sum_s \tau_s 2^{-i} \psi(2^{-i}s - t),\ i = 1, \ldots, L, \\ a_t^L &= \sum_s \tau_s 2^{-L} \phi(2^{-L}s - t), \end{aligned}$$

and $\psi(.)$ is a mother wavelet function and $\phi(.)$ its corresponding scaling functions [14].

Now, an anomaly detection mechanism, similar to that described in the basic analysis using variance subsection, is applied to each details time series. For each level $l \in [1, L]$we create a 0-1 sequence: each time instant t is assigned either a 0 or 1 where 0 indicates that no anomaly was detected and 1 means an anomaly was flagged. By summing across these 0-1 time series, for a given time instant, we have the number of times that an anomaly was detected across all the details signals. The larger this numer, the more time scales at which the anomaly was detected. (In practice, we sum not over a single time instant, but over a small window in each signal). An anomaly flag is raised if the anomaly is detected at a sufficient number of scales. The computation of the wavelet introduces a lag in the detection; this lag will be a function of of the largest scale used.

3.5 Multi scale variance shift

This method is derived from [2]. In this paper the authors detect the difference between the local and the global variance of the process. They first remove the trend of the signal using a wavelet transform, *i.e.* the remove the approximation part of a wavelet transform. Thereafter they use a small window to compute a local variance. Whenever the ratio between this local variance and the global variance (computed on all the data) exceeds a threshold T then an alarm is triggered.

This method is in fact a special case of the multiscale analysis previously described, where only two scales are analyzed, the scale at which the global variance is calculated and the local scale where the local variance is calculated. The approach can be assimilated to wavelet transform with a Haar wavelet. The other interesting point of the approach is that it detects a variation in the variance of the process in place of detecting a variation in the mean as previously described approaches. It is noteworthy that other approaches could also be adapted to detecting changes in variance in place of the mean.

This method will also experience a detection lag time, since the wavelet approach introduces a lag due to the time needed to compute the wavelet transform in the two scales. The width of the window of time over which to computes the local variance is very important and will depend on the duration of the anomaly to detect.

4 Validation Methodology

The validation of any anomaly detection method is always fraught with difficulty. The challenge comes from our inability to establish the ”ground truth”. Among the most interesting performance metrics for such methods are the false positive and false negative rates. However computing these rates requires us to know exactly which events (and corresponding point in time) were anomalies and which were not. One common approach to evaluating anomaly detection algorithms is to collect live data in the form of a packet or flow level trace, and then to have this trace ”labeled”. Labeling or marking a trace is the procedure by which each anomalous event is identified along with its start and finish time. Perhaps the best way to do this in today’s world is for a security operations expert to do the labeling either via visual inspection or with the help of tools. They have a wealth of real world experience that is hard to automate. Although this is currently our best option, the labeling method is not perfect as operators can make mistakes, either missing an anomaly or generating a false positive. The advantage of using labeled traces is that they capture real world events. The disadvantage is that such traces contain a fixed number of events whose parameters cannot be varied. For example, one cannot ask ”suppose the volume of the attack had been a little lower, would our algorithm have caught it?”

A second approach to validation is to synthetically generate attacks. The advantage of this approach is that the parameters of an attack (attack rate, duration, number of flows involved, etc.) can be carefully control. One can then attempt to answer the above question. This enables sensitivity testing of any detection algorithm. Clearly the disadvantage is that these attacks have not happened anywhere and thus may be of less interest.

We believe that a good approach to validation of an anomaly detection algorithm should contain both of the above approaches, so as to obtain the benefits of each method. For our set of real world data with anomalies, we obtained four weeks of traffic matrix data from the Abilene Internet2 backbone network. Abilene is a major academic network, connecting over 200 US universities and peering with research networks in Europe and Asia. This data was labeled using the method in [12]. We developed our own synthetic anomaly generator and implemented it in Matlab. This is described in detail further below.

4.1 Abilene Data

The Abilene backbone has 11 Points of Presence (PoP) and spans the continental US. The data from this network was collected from every PoP at the granularity of IP level flows. The Abilene backbone is composed of Juniper routers whose traffic sampling feature was enabled. Of all the packets entering a router, 1% are sampled at random. Sampled packets are aggregated at the 5-tuple IP-flow level and aggregated into 5 minute bins. This thus dictates the underlying time unit of all of our estimations and detections. The raw IP flow level data is converted into a PoP-to-PoP level matrix using the procedure described in [11]. Since the Abilene backbone has 11 PoPs, this yields a traffic matrix with 121 OD flows. Note that each traffic matrix element corresponds to a single OD flow, however, for each OD flow we have a four week long time series depicting the evolution (in 5 minute increments) of that flow over the measurement period.

4.2 Synthetic Anomaly Generation

Our approach to synthetically generation anomalies makes use of the Abilene traffic matrix. The idea is to select either one, or a set of, OD flows to be involved in the anomaly, and then to add anomalies on top of the baseline traffic level for those OD flows. Our reasons for adding anomalies on top of the existing traffic matrix are as follows. We want to detect anomalies using the ensemble of all network links, we need to populate the load of the entire network. Other methods are available such as [17]. But they generate single link packet traces while we need multi-link SNMP data. Thus we rely on our measured dataset for generating malicious data.

Using the abilene traffic matrix allows us to recreate realistic loads network-wide. This includes all the many sources of variability exhibited on the set of network links.

To inject an anomaly into this network, we use a three step procedure. These procedure is carried out for each OD flow involved in the anomaly.

1. Extract the long-term statistical trend from the selected OD flow. The goal is to capture the diurnal pattern by smoothing the original signal.

2. Add Gaussian noise onto the smoothed signal.

3. Add one of the anomalies as described in Table 1 on top of this resulting signal.

These three steps are depicted pictorially in Figure 2. It was shown in [19] that OD pairs in an ISP exhibit strong diurnal patterns. These 24-hour cycles represent normal types of variability in aggregated traffic. Another normal source of variability in OD flows simply comes from noise [13], and thus the first two steps are intended to represent the level of traffic in an OD flow right before the anomaly starts; this should look like regular non-anomalous traffic.

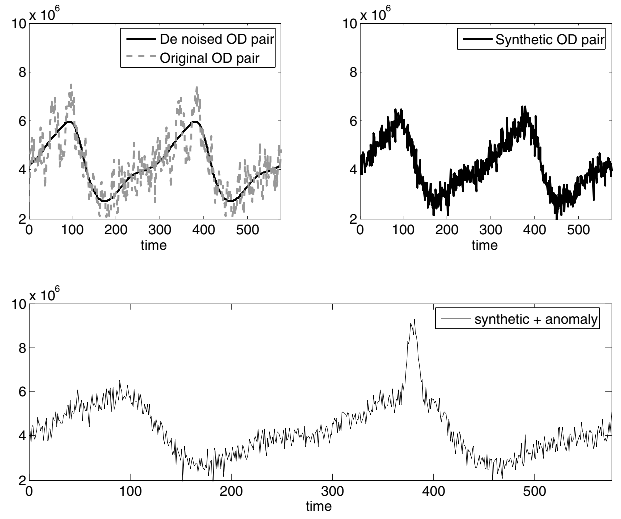


Figure 2: Three steps for synthetic generation of an anomaly.

We extract the diurnal trend using a discrete wavelet transform; wavelet methods here useful since these trends are typically non-stationary. Evidence of the ability of spectral methods to capture the underlying trends in highly aggregated traffic has been observed in [19, 8, 2]. We compute the first five approximation signals using a Daubechies-5 mother wavelet with 5 levels. We keep the approximation signal at the 5th level, thus filtering out everything except this smoothed signal. This smoothed, or de-noised, signal is shown in the top left plot of Figure 2 as the solid line. We add to this baseline signal a zero mean Gaussian noise who variance is computed as follows. We take the first 5 detailed signals from our wavelet transform, and compute the variance of the sum of the 5 detailed signals. A sample signal produced after step 2 is depicted in the upper right plot of Figure 2. An important reason to use a signal that has been smoothed and only supplemented with Gaussian noise is to ensure that there is no anomaly in this OD flow other than the one we are about to add.

The last step is to add an anomaly onto this baseline traffic. This is depicted in the bottom plot of Figure 2 where we see the anomaly added on top of the filtered OD flow. In our synthetic anomaly generator we characterize each anomaly by four parameters, namely, *volume*, *duration*, *number of OD flows involved*, and a *shape* function. The *shape* function refers to the rate of increase when the anomaly begins (also called *ramp up*), as well as the rate of decrease as the anomaly tapers off. We include four different shape functions: ramp, exponential, square and step. The ramp function is further characterized by a slope parameter, and the exponential shape by its rate parameter. Our intent is to define a feasible range for each of these parameters such that we are able to capture the general behavior of known anomaly types as well as to encompass a broader range of behaviors.

As pointed out in [15], there are unfortunately no comprehensive studies yet that provide detailed statistical descriptions of a broad set of volume anomalies. There are a handful of studies [2, 11, 16, 9, 3] that provide useful pieces of information towards this end. The characterization part of these studies often touch briefly on a wide variety of metrics, from attack rate and duration to others such as the distribution of source or victim IP addresses, type of protocol involved in the attack, and the effect on the end system (e.g., number of sessions open), etc. Some of these studies do provide a few statistics on the parameters we wish to calibrate. Whenever possible, we draw upon these works and include their findings as particular examples. As it is hard to generalize from these specific cases, we allow our parameters to vary through a broader range than those found in these studies.

The types of anomalies we would like to be able to mimic include: DDOS, flash crowd, alpha, outages and ingress/egress shift. Since we focus on detecting changes in traffic volume patterns, we do not include other anomalies such as worms and scans. A *DDOS* attack represents a flooding attack against a single destination. These attacks can have either a single source (DOS) or many distributed sources (DDOS). The latter occurs when many machines (called 'zombies') are compromised and a single attacker sends commands to all of the zombies enabling them to jointly flood a victim. A *flash crowd* occurs when there is a surge in demand for a service and is typically manifested by a large number of clients trying to access, and thus overwhelming, a popular Web site. Flash crowds can be predictable (e.g. a scheduled baseball game, or a software release) or unpredictable (e.g., news breaking event) [9].

An *alpha* anomaly refers to the transfer of a file(s) with an unusually large number of bytes. This typically involves one OD flow as there is a single source and a single destination. An *outage* refers to scenarios such as failures which can cause the load on a link to drop to zero. Such drops can either be short-lived or long-lived, and the short-lived outages are not infrequent since failures of one sort or another are fairly commonplace in the Internet today [1]. An *egress shift* occurs when the destination of an OD flow moves from one node to another. This can happen in a traffic matrix if there is a change in a BGP peering policy, or even a failure, as many OD flows can have multiple possible exit points from an ISP. Policy changes could also cause a shift of ingress point for a particular destination. In [21] the authors showed that traffic movement due to ingress or egress shifts, although not frequent, does indeed happen. None of these anomalies, other than DDOS attacks, are malicious. Yet all of them will generate potentially sudden and large shifts in traffic patterns, thus appearing anomalous.

In Table 1 we list our five parameters characterizing an anomaly. For each parameter we list the options for values, or value ranges, that the parameter can take on. We allow the duration to be anything from minutes, to hours, to days and for forever. We include the forever case as this includes the ingress and egress shift anomalies that will last until there is another policy change. Since [21] indicates these events are not that frequent, we can view the shift in traffic pattern as "permanent". The duration of an anomaly can vary throughout a large range, and it is unclear what the future will bring. Although most DDOS attacks observed, in the backscatter data of [16], lasted between 5 and 30 minutes, there were some outliers lasting less than 1 minute and others that lasted several days. Similarly, the majority of the DDOS events in the Abilene data of [11], lasted less than 20 minutes; a few outliers exceeds 2 hours. Alpha and flash crowd events could be of any length, although typically alpha events would be shorter than flash crowd events. In general, we do not include events whose order or magnitude of duration are less than minutes because we are adding these events on top of the Abilene data that is available to us with a minimum time interval of 5 minutes.

We change the traffic volume in two ways when anomalies occur. Sometimes we use a multiplicative factor δ that is multiplied by the baseline traffic to generate the new traffic load. Using $\delta \sim 0$, we can easily capture outage scenarios. When an egress shift occurs, we assume that a subset of the prefixes travelling between the source and destination router are being shifted to a new exit point. This will shift a portion of the router-to-router traffic (as these policies are more likely to affect only a subset of the IP level prefixes) from the old OD pair to the new one. Removing 10%, for example, of the original OD flow's data is simply captured by using $\delta = 0.9$. This amount of traffic is added into the new OD flow using the constant additive term Δ. Allowing $1 \leq \delta \leq 2$, we can capture a variety of either alpha, flash crowd or DOS events. Note that because we are considering aggregate flows at the router to router level, doubling the traffic from an ingress router is already an enormous increase in traffic load. Large increases can occur when there are many end hosts behind the router that are involved in the anomaly (e.g., zombies, flash crowd). We don't consider $\delta > 2$ because such attacks are so obviously irregular that they are trivial to detect. We also allow a change in volume to be indicated by simply adding a constant factor, Δ, into the existing volume. This can capture the effect of a DDOS attack in which many zombies flood a victim at their maximum rate.

The *number of sources and destinations* indicates the number of OD flows involved in an anomaly. The notation $(1, 1)$ refers to a single source and a single destination. This could happen either for a DOS attack or an alpha event. The case of $(N, 1)$ arises for DDOS and flash crowds. In the case of a link failure, all the OD flows traversing the link are affected. The case of $(2, 2)$ can occur for an ingress or egress shift. By this we mean that there are two OD flows involved (that share either a common source or destination). One of these flows will experience an increase in volume, while the other experiences an equal amount of decrease. We do not include the case of (k, k) because we assume that one BGP policy will change at a time.

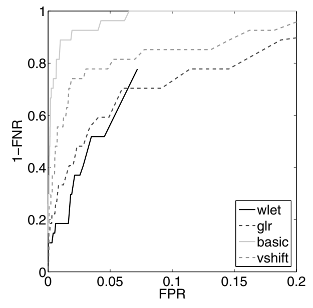
As mentioned earlier, our shape function can take on one of four possible forms: a ramp, exponential, square or step function. The shape function is multiplied by the extra volume amount before it is added onto the baseline traffic. This thus determines the ramp up and drop-off behavior of most anomalies. Not only are these shapes intuitively useful, but there is also some evidence for them in existing datasets. In [3] the authors found that a flash crowd can be characterized by a rapid rise in traffic that is then followed by a gradual drop-off over time. It also has been shown for flash crowd events that although their ramp up can be very quick, it is typically not instantaneous [9]. The initial increase of a DDOS attack could be captured by a ramp; this allows us the flexibility of representing scenarios in which the zombies reach their maximum flood rates in succession (medium slope) or via a very sharp rise [3] (steep slope). Outage anomalies could exhibit a near instantaneous drop in volume and thus we include the 'square' function. Alpha events could exhibit either a near instantaneous increase in volume or a ramp up. The step function is included to represent the ingress or egress shift anomalies because in these cases the change in traffic pattern is permanent (at least until the next policy change).

When we generate an anomaly we randomly select the values for these four parameters. Some combinations of them will look like the anomalies we have discussed. By varying each of the four characteristics in our generator, we can create a wide variety of anomalies.

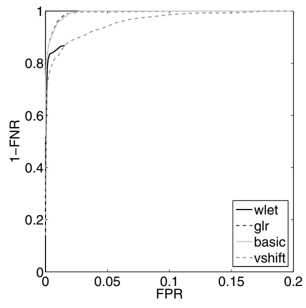
5 Results

5.1 False Positive and False Negative Performance

We start by looking at the performance of our methods in the Abilene network. The abilene data contains 27 anomalies. Within each method, for each value of the threshold, we examine the entire traffic matrix (thus traversing all anomalies and non-anomalies). We can thus compute one false positive percentage and one false negative percentage for each threshold configuration of a scheme. The performance of our 4 methods on the Abilene data is depicted in the ROC curve of Figure 3(a). We see clearly that the basic method performs best. For a false positive rate of 7%, it misses no anomalies (100% true positives), while the next best method catches about 85% of the true anomalies for the same false positive rate. The wavelet method was unable to achieve 0% false negatives. Thus we observe an incomplete curve that does not reach the FNR $= 0$ limit, even with a huge threshold.



(a) Abilene



(b) Synthetic

Figure 3: ROC curves using Abilene and Synthetic data

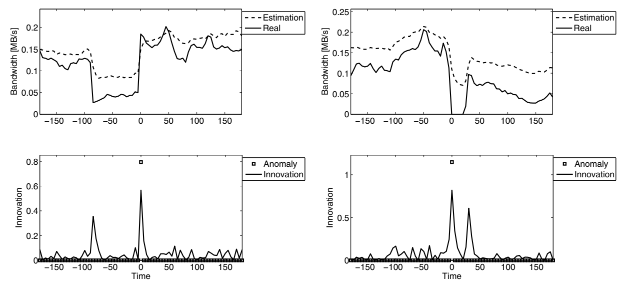
We now examine the performance of our algorithms using our synthetic anomaly generator. We generated about 500 different anomalies by varying the parameters of our generator. For these attacks, the duration was varied ran-

Parameter	Duration	Volume	Num (Src,Dst)	Shape
possible values	Minutes	Δ	$(1, 1)$	Ramp
	Hours	$1 \leq \delta \leq 2$	$(N, 1)$	Exponential
	Days	$\delta = 1.1$ or 0.9	$(2, 2)$	Square
	Forever	$\delta \sim 0$	all ODs on 1 link	Step

Table 1: Anomaly description parameters. Δ is an additive factor, δ is a multiplicative factor.

domly between 5 min and 2 hours. The volume of the original OD flow added on top of the anomalous OD pair ranged between 40% and 140%. The number of OD pairs involved was between 1 and 7 OD pairs per anomaly, and those selected were randomly chosen. The performance of our four schemes for these 500 scenarios is presented in Figure 3(b). For this data, the basic and GLR performed best and equivalently. It is interesting to note that the ranking of the four schemes, in terms of the ROC curve areas is not entirely consistent between the Abilene data and the synthetic ones. The main difference occurs with the GLR method that does not perform very well for the Abilene data but does for the synthetic data. The reason may lie in the statistical properties of the anomalies themselves. In our synthetic generator the way we add extra volume is equivalent to changing the mean of the OD flow for the duration of the anomaly. Since the GLR method is focused on detecting changes in the mean, it does well. It is possible that the anomalies in the Abilene data experience variance changes as well as mean changes. If this were true, it would explain why the vshift method is second best for the Abilene data. We leave the exploration of the statistical properties of the anomalous moments for future work.

When using marked traces we should be careful. There is always the risk that an anomaly is undetected or a normal behavior is marked as an anomaly. We conducted a visual inspection to remove any false positive(s) detected using the algorithms presented in [12]. We did not check for the false negatives. Consider the examples in Figures 4(a) and 4(b). On the top plots we show how a single OD pair evolves in time. The dashed line is our kalman filter estimation of this OD flow. We can see how it tracks the changes in the OD flow. On the bottom plots we show the residual process for each of these two example flows. We also include the markings produced by the labeling algorithm in [12]. Each box greater than 0 means that an anomaly was marked at this time. In figure 4(a) our residual process indicates that there were two anomalies, while the labeling procedure only marks one of them. According to our methodology above, we would thus label the first spike as a false positive since we use the labeling method to represent the "truth". This anomaly could easily have been a legitimate one. A similar situation arises for our second example flow. For these two examples, a simple



(a) OD pair #19 (b) OD pair #32

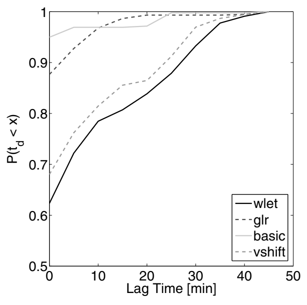
Figure 4: Example of the Innovation of an OD pair

visual inspection of the upper curve is enough to indicate that these events should have been True Positives since the two anomalies per flow indicate the beginning and ending of the anomaly. Because our algorithm may be able to detect events that the labeling algorithm of [12] does not, yet we use this algorithm to compute the FP ratio, it means that our computed false positive rate should be considered as an upper bound instead of the true value of the false positive ratio.

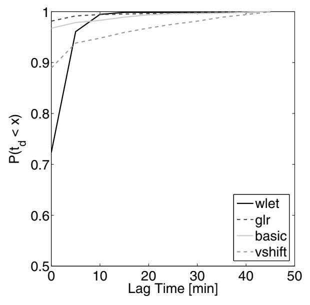
5.2 Detection Time

One of the critical performance aspects of any anomaly detection algorithm is the speed with which it can detect the anomaly. The onset of attacks and/or anomalies on the Internet today is extremely rapid thus creating real-time requirements for anomaly detection algorithms that are challenging. Few, if any, of the previous work we have seen, evaluate their algorithms in terms of detection time. We define detection lag as the time at which we detect a true anomaly minus the time the anomaly began. Since the underlying time unit of our traffic matrix data is 5 minutes, each additional lag corresponds to an increment of 5 minutes. (Note that our methods are not intrinsically tied to a 5 minute time interval.)

Each anomaly in the two sets (Abilene and synthetic) generates one sample detection lag value. We ensemble all these values and summarize them using a cumulative distribution. The results for the Abilene data are shown in Fig-

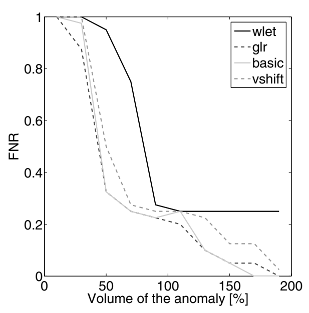


(a) Abilene

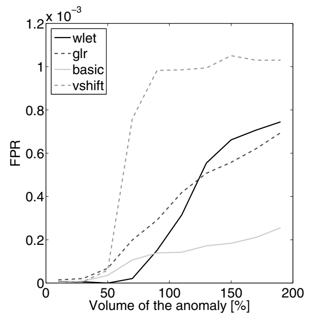


(b) Synthetic

Figure 5: CDF of the detection lag using Abilene and Synthetic traces



(a) FNR



(b) FPR

Figure 6: FNR and FPR as a function of the anomaly size

ure 5(a) while the results for the synthetic data are shown in Figure 5(b). In both cases, the basic method and GLR methods exhibit excellent detection times. In the case of Abilene data, the GLR method detected 90% of the anomalies with no lag, while the basic method detected 95% of the anomalies with no lag. For the synthetic data, the GLR curve is not visible because it lies on the line where the y-axis is 1 (underneath the basic curve). For the synthetic cases, both the GLR and basic methods were able to detection 100% of the anomalies with no lag at all. The wavelet analysis method performs less well; in particular there appear to be some difficult anomalies that can take over half an hour to detect. It is interesting that the vshift method performs well for the synthetic data but not for the Abilene data. In the synthetic case to detect an anomaly the volume should be high enough to raise an alarm as soon as it is observed otherwise it remains undetected and we cannot computes a lag time. Whereas in the Abilene data the vshift method is able to detect a subtle deviation in the statistics of the process and therefore need more samples to detect it. The motivation for using a wavelet method was an intuition that "an anomaly should diffuse itself at several time scales". However, in the results the anomalies appear differently at different time scales, and hence this approach was not very powerful in detecting anomalies. Other uses of wavelet methods in this context might prove more beneficial. For example, they might be useful for classifying anomalies since wavelet methods can give a rich description of the anomaly dynamics. This interesting problem is out of the scope of this paper.

5.3 Sensitivity Analysis

It is intuitive that enormous anomalies will be easy to detect and that very tiny ones are going to be missed. It is interesting to explore the space in between and see the impact of the false positive and false negative ratios as the volume of anomalies get smaller and smaller. In Figure 6(a) we plot the false negative ratio versus the percentage increase in the anomalous flow. To get a broad range of anomalies, for each tested volume level, we generate 50 anomalies with various start times or number of OD flows involved. We did this for 10 different volumes with $1.2 \leq \delta \leq 2$.

Figure 6(a) matches our intuition. If the OD flow increases by only 10 or 20% of its original value, we going to miss the anomalies. However the drop off of three methods is similar and fairly quick in the range of 40 - 100%. This implies that if the load from an ingress node doubled, it should be easy to catch all anomalies (low missed anomaly rate). Note that this justifies the fact that we don't use $\delta > 2$ in our synthetic anomaly generator.

The curve for the false positive rate (Figure 6(b)) is surprising. Initially we would have expected for this also to be a decreasing curve. But, as the anomaly becomes larger ($\delta \geq 1.5$) all the flows sharing a link have their estimates corrected by a large amount. Thus the error is spread inside the kalman filter to normal OD flows. This in turn increases the innovations leading to more false positives. This will not impact the ability to detect an anomaly but rather cloud the identity of the OD flow carrying the anomaly.

6 Conclusions

Our solution to tackling volume anomalies in large networks consists of many parts. First we select an interesting granularity level at which to perform anomaly detection, namely that of a traffic matrix. Second we use kalman filters to filter the predictable part of the traffic and to isolate the prediction error. The form of our model allows us to obtain the prediction error on the unobservable part of the network system (the OD flows) as well as for the observable part (link loads). Third, we proposed two detection schemes, but compared the performance of four of them. Finally we discuss how to make decisions about the presence of anomalies through the use of statistical hypothesis testing. We argue that the main measure of performance of an anomaly detector should be the ROC curve that explicitly captures the relationship between false positive and false negative rates. We give a mathematical foundation for this approach through the Neyman-Pearson theorem that identifies how to select decision thresholds when balancing the false positive and false negative tradeoff.

We considered four detection schemes that differ in the statistical change they seek to detect. Interestingly, but perhaps not surprisingly in retrospect, we found that the GLR method (whose goal is to detect changes in the mean) performs best when the anomaly is one that causes a change in the mean (e.g., in the synthetic cases). Similarly we found that the 'vshift' method performs better for the Abilene data than the synthetic data. We hypothesize that this occurs because the statistical properties of the anomalies themselves in the Abilene data contain changes in the variance of the residual traffic process. (We intend to verify this in future work by adding extra features into our synthetic anomaly generator that will alter the variance of the anomaly.) If the latter hypothesis is true, the implication is that the statistical change method that works best is the one checking the parameter that undergoes a deviation in the anomaly. On the one hand, this is motivation to do a study of the statistical properties of anomalies themselves. On the other hand, it suggests that the best method for network administrators could be a composite method that makes use of multiple different kinds of tests.

In our study, the wavelet based method did not perform well. Due to the popularity of wavelet based analyses, this raises interesting questions as to when wavelet analysis is and isn't useful for the problem domain of anomaly detection. Most importantly, from a practical point of view, it is good news that the simplest method performed best across all validation tests. This could be due to the fact that the Kalman model for the OD flows correctly models the normal traffic and thus the first filtering step is successful itself in isolating anomalies.

Aknowledgements

We would like to thank Anukool Lakhina for generously sharing his labeled Abilene traces with us. We are also grateful to Simon Crosby for talking to us about ROC curves.

References

[1] A.MARKOPOULOU, IANNACCONE, G., BHATTACHARYYA, S., CHUAH, C., AND DIOT, C. Characterization of Failures in an IP Backbone. In *In: IEEE Infocom* (March 2004).

[2] BARFORD, P., KLINE, J., PLONKA, D., AND RON, A. A signal analysis of network traffic anomalies. *ACM Sigcomm IMW* (2002).

[3] BARFORD, P., AND PLONKA, D. Characterisitics of of network traffic flow anomalies. In *ACM IMW* (Nov. 2001).

[4] BASSEVILLE, M., AND NIKIFOROV, I. Detection of abrupt changes: theory and application, 1993.

[5] EGAN, J. *Signal Detection Theory and ROC Analysis*. Academic Press, 1975.

[6] GUNNAR, A., JOHANSSON, M., AND TELKAMP, T. Traffic matrix estimation on a large ip backbone - a comparison on real data. In *ACM IMC* (Oct. 2004).

[7] HAWKINS, D. M., QQUI, P., AND KANG, C. W. The changepoint model for statistical process control. *Journal of Quality Technology 35*, 4 (october 2003).

[8] HUSSAIN, A. *Measurement and Spectral Analysis of Denial of Service Attacks*. PhD thesis, USC, May 2005.

[9] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *ACM WWW Conference* (May 2002).

[10] KAILATH, T., SAYED, A. H., HASSIBI, B., SAYED, A. H., AND HASSIBI, B. *Linear Estimation*. Prentice Hall, 2000.

[11] LAKHINA, A., CROVELLA, M., AND DIOT., C. Characterization of network-wide anomalies in traffic flows. In *ACM IMC* (2004).

[12] LAKHINA, A., CROVELLA, M., AND DIOT., C. Diagnosing network-wide traffic anomalies. In *ACM Sigcomm* (2004).

[13] LAKHINA, A., PAPAGIANNAKI, K., CROVELLA, M., DIOT, C., KOLACZYK, E., AND TAFT, N. Structural analysis of network traffic flows. In *ACM Sigmetrics* (2004).

[14] MALLAT, S. *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[15] MIRKOVIC, J., AND REIHER, P. A taxonomy of ddos attack and ddos defense mechanisms. In *ACM CCR* (April 2004).

[16] MOORE, D., VOELKER, G. M., AND SAVAGE, S. Inferring internet Denial-of-Service activity. In *Proceedings of the 10th USENIX Security Symposium* (2001), pp. 9–22.

[17] SOMMERS, J., YEGNESWARAN, V., AND BARFORD, P. A framework for malicious workload generation. In *IMC* (New York, NY, USA, 2004), ACM Press, pp. 82–87.

[18] SOULE, A., LAKHINA, A., TAFT, N., PAPAGIANNAKI, K., SALAMATIAN, K., NUCCI, A., CROVELLA, M., AND DIOT, C. Traffic matrices: Balancing measurements, inference and modeling. In *ACM Sigmetrics* (2005), ACM Press.

[19] SOULE, A., NUCCI, A., CRUZ, R., LEONARDI, E., AND TAFT, N. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *ACM Sigmetrics* (New York, 2004).

[20] SOULE, A., SALAMATIAN, K., AND TAFT, N. Traffic matrix tracking using kalman filters. *ACM LSNI Workshop* (2005).

[21] TEIXEIRA, R., DUFFIELD, N., REXFORD, J., AND ROUGHAN, M. Traffic matrix reloaded: Impact of routing changes. In *PAM* (2005).

[22] ZHANG, Y., ROUGHAN, M., DUFFIELD, N., AND GREENBERG, A. Fast accurate computation of large-scale ip traffic matrices from link loads. In *ACM Sigmretrics* (2003), ACM Press, pp. 206–217.

[23] ZHANG, Y., ROUGHAN, M., LUND, C., AND DONOHO, D. An information-theoretic approach to traffic matrix estimation. In *ACM Sigcomm* (2003), ACM Press, pp. 301–312.

[24] ZWEIG, M. H., AND CAMPBELL, G. Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. In *Clinical Chemisty* (1993), vol. 93(4).

Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation

Yu Gu, Andrew McCallum, Don Towsley
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003

Abstract

We develop a behavior-based anomaly detection method that detects network anomalies by comparing the current network traffic against a baseline distribution. The Maximum Entropy technique provides a flexible and fast approach to estimate the baseline distribution, which also gives the network administrator a multi-dimensional view of the network traffic. By computing a measure related to the relative entropy of the network traffic under observation with respect to the baseline distribution, we are able to distinguish anomalies that change the traffic either abruptly or slowly. In addition, our method provides information revealing the type of the anomaly detected. It requires a constant memory and a computation time proportional to the traffic rate.

1 Introduction

Malicious abuses of the Internet are commonly seen in today's Internet traffic. Anomalies such as worms, port scans, denial of service attacks, etc. can be found at any time in the network traffic. These anomalies waste network resources, cause performance degradation of network devices and end hosts, and lead to security issues concerning all Internet users. Thus, accurately detecting such anomalies has become an important problem for the network community to solve.

In this paper, we develop a network anomaly detection technique based on maximum entropy and relative entropy techniques. Our approach exploits the idea of behavior-based anomaly detection. We first divide packets into classes along multiple dimensions. A maximum entropy *baseline distribution* of the packet classes in the benign traffic is determined by learning a density model from a set of pre-labeled training data. The empirical distribution of the packet classes under observation is then compared to this baseline distribution using relative entropy as the metric. If the two distributions differ, we show that the packet classes primarily responsible for the difference contain packets related to an anomaly.

The maximum entropy approach described in this work exhibits many advantages. First, it provides the administrators a multi-dimensional view of the network traffic by classifying packets according to a set of attributes carried by a packet. Second, it detects anomalies that cause abrupt changes in the network traffic, as well as those that increase traffic slowly. A large deviation from the baseline distribution can only be caused by packets that make up an unusual portion of the traffic. If an anomaly occurs, no matter how slowly it increases its traffic, it can be detected once the relative entropy increases to a certain level. Third, it provides information about the type of the anomaly detected. Our method requires only a constant amount of memory and consists solely of counting the packets in the traffic, without requiring any per flow information.

Our approach divides into two phases. Phase one is to learn the baseline distribution and phase two is to detect anomalies in the observed traffic. In the first phase, we first divide packets into multi-dimensional packet classes according to the packets' protocol information and destination port numbers. These packet classes serve as the domain of the probability space. Then, the baseline distribution of the packet classes is determined by learning a density model from the training data using Maximum Entropy estimation. The training data is a pre-labeled data set with the anomalies labeled by a human and in which packets labeled as anomalous are removed. During the second phase, an observed network traffic trace is given as the input. The relative entropy of the packet classes in the observed traffic trace with respect to the baseline distribution is computed. The packet classes that contribute significantly to the relative entropy are then recorded. If certain packet classes continue to contribute significantly to the relative entropy, anomaly warnings are generated and the corresponding packet classes are reported. This corresponding packet class information reveals the protocols and the destination port numbers related to the anomalies.

We test the approach over a set of real traffic traces. One of them is used as the training set and the others are used as the test data sets. The experimental results show that our approach identifies anomalies in the traffic with low false negatives and low false positives.

The rest of the paper is organized as follows. In Section 2, we review related work. Section 3 describes how we classify the packets in the traffic. In Section 4, we introduce the Maximum Entropy estimation technique. In Section 5, we describe how to detect anomalies in the network traffic based on the baseline distribution. Section 6 gives experimental results and Section 7 discusses the implementation of the algorithm and related practical issues. The last section summarizes the whole paper.

2 Related work

A variety of tools have been developed for the purpose of network anomaly detection. Some detect anomalies by matching the traffic pattern or the packets using a set of predefined rules that describe characteristics of the anomalies. Examples of this include many of the rules or policies used in Snort [12] and Bro [10]. The cost of applying these approaches is proportional to the size of the rule set as well as the complexity of the individual rules, which affects the scalability of these approaches. Furthermore they are not sensitive to anomalies that have not been previously defined. Our work is a behavior based approach and requires little computation.

A number of existing approaches are variations on the change detection method. In [2], Brutlag uses the Holt Winter forecasting model to capture the history of the network traffic variations and to predict the future traffic rate in the form of a confidence band. When the variance of the network traffic continues to fall outside of the confidence band, an alarm is raised. In [1], Barford *et al.* use wavelet analysis to remove from the traffic the predictable ambient part and then study the variations in the network traffic rate. Network anomalies are detected by applying a threshold to a deviation score computed from the analysis. In [14], Thottan and Ji take management information base (MIB) data collected from routers as time series data and use an auto-regressive process to model the process. Network anomalies are detected by inspecting abrupt changes in the statistics of the data. In [15], Wang *et al.* take the difference in the number of SYNs and FINs (RSTs) collected within one sampling period as time series data and use a non-parametric Cumulative Sum (CUSUM) method to detect SYN flooding by detecting the change point of the time series. While these methods can detect anomalies that cause unpredicted changes in the network traffic, they may be deceived by attacks that increase their traffic slowly. Our work can detect anomalies regardless of how slowly the traffic is increased and report on the type of the anomaly detected.

There is also research using approaches based on information theory. In [7], Lee and Xiang study several information theoretic measures for intrusion detection. Their study uses entropy and conditional entropy to help data partitioning and setting parameters for existing intrusion detection models. Our work detects network traffic anomalies that cause unusual changes in the network traffic rate or content. In [13], Staniford *et al.* use information theoretic measures to help detect stealthy port scans. Their feature models are based on maintaining probability tables of feature instances and multi-dimensional tables of conditional probabilities. Our work applies a systematic framework, Maximum Entropy estimation, to estimate the baseline distribution, and our approach is not limited to locating port scans.

Maximum Entropy estimation is a general technique that has been widely used in the fields of machine learning, information retrieval, computer vision, and econometrics, etc. In [11], Pietra *et al.* present a systematic way to induce features from random fields using Maximum Entropy technique. In [9], McCallum builds, on [11], an efficient approach to induce features of Conditional Random Fields (CRFs). CRFs are undirected graphical models used to calculate the conditional probability of values on designated output nodes given values assigned to other designated input nodes. And in [8], Malouf gives a detailed comparison of several Maximum Entropy parameter estimation algorithms. In our work, we use the L-BFGS algorithm implemented by Malouf to estimate the parameters in the Maximum Entropy model.

3 Packet classification

In this section, we describe how we divide packets in the network traffic into a set of packet classes. Our work focuses on anomalies concerning TCP and UDP packets. In order to study the distribution of these packets, we divide them into a set of two-dimensional classes according to the protocol information and the destination port number in the packet header. This set of packet classes is the common domain of the probability spaces in this work.

In the first dimension, packets are divided into four classes according to the protocol related information. First, packets are divided into the classes of TCP and UDP packets. Two other classes are further split from the TCP packet class according to whether or not the packets are SYN and RST packets.

In the second dimension, packets are divided into 587 classes according to their destination port numbers. Port numbers often determine the services related to the packet exchange. According to the Internet Assigned Numbers Authority [6], port numbers are divided into three categories: *Well Known Ports* ($0 \sim 1023$), *Registered Ports* ($1024 \sim 49151$), and *Dynamic and/or Private Ports*

($49152 \sim 65535$). In our work, packets with a destination port in the first category are divided into classes of 10 port numbers each. Since packets with port number 80 comprise the majority of the network traffic, they are separated into a single class. This produces 104 packet classes. Packets with destination port in the second category are divided into 482 additional classes, with each class covering 100 port numbers with the exception of the class that covers the last 28 port numbers from 49124 to 49151. Packets with destination port numbers larger than 49151 are grouped into a single class. Thus, in this dimension, packets are divided into a total of $104 + 482 + 1 = 587$ classes.

Altogether, the set of two-dimensional classes consists of $4 * 587 = 2348$ packet classes. These packet classes comprises the probability space in this paper. We estimate the distribution of different packets in the benign traffic according to this classification, and use it as the baseline distribution to detect network traffic anomalies.

4 Maximum Entropy estimation of the packet classes distribution

Maximum Entropy estimation is a framework for obtaining a parametric probability distribution model from the training data and a set of constraints on the model. Maximum Entropy estimation produces a model with the most 'uniform' distribution among all the distributions satisfying the given constraints. A mathematical metric of the uniformity of a distribution P is its entropy:

$$H(P) = -\sum_{\omega\in\Omega} P(\omega)\log P(\omega). \quad (1)$$

Let Ω be the set of packet classes defined in the previous section. Given a sequence of packets $\mathcal{S} = \{x_1, \ldots, x_n\}$ as the training data, the empirical distribution $\tilde{P}$ over Ω in this training data is

$$\tilde{P}(\omega) = \frac{\sum \mathbf{1}(x_i \in \omega)}{n}, \quad (2)$$

where $\mathbf{1}(X)$ is an indicator function that takes value 1 if X is true and 0 otherwise.

Suppose we are given a set of feature functions $\mathcal{F} = \{f_i\}$, and let f_i be an indicator function $f_i : \Omega \mapsto \{0,1\}$. By using Maximum Entropy estimation, we are looking for a density model P that satisfies $E_P(f_i) = E_{\tilde{P}}(f_i)$ for all $f_i \in \mathcal{F}$ and has maximum entropy. In [11], it has been proved that under such constraints, the Maximum Entropy estimate is guaranteed to be (a) unique, and (b) the same as the maximum likelihood estimate using the generalized Gibbs distribution, having the following log-linear form

$$P(\omega) = \frac{1}{Z}\exp(\sum_i \lambda_i f_i(\omega)). \quad (3)$$

For each feature f_i, a parameter $\lambda_i \in \Lambda$ determines its weight in the model, Λ is the set of parameters for the feature functions. Z is a normalization constant that ensures that the sum of the probabilities over Ω is 1. The difference between two given distributions P and Q is commonly determined using the *relative entropy* or *Kullback-Leibler (K-L) divergence*:

$$D(P||Q) = \sum_{\omega\in\Omega} P(\omega)\log\frac{P(\omega)}{Q(\omega)}.$$

Maximizing the likelihood of the distribution in the form of (3) with respect to $\tilde{P}$ is equivalent to minimizing the K-L divergence of $\tilde{P}$ with respect to P

$$P = \arg\min_P D(\tilde{P}\|P)$$

as

$$\prod_{\omega\in\Omega} P(\omega)^{\sum \mathbf{1}(x_i\in\omega)} \propto \exp(-D(\tilde{P}\|P)).$$

For the sake of efficiency, feature functions are often selected to express the most important characteristics of the training data in the learned log-linear model, and in return, the log-linear model expresses the empirical distribution with the fewest feature functions and parameters.

The Maximum Entropy estimation procedure consists of two parts: feature selection and parameter estimation. The feature selection part selects the most important features of the log-linear model, and the parameter estimation part assigns a proper weight to each of the feature functions. These two parts are performed iteratively to reach the final model. In the following, we describe each part in turn. More details can be found in [11].

4.1 Feature selection

The feature selection step is a greedy algorithm which chooses the best feature function that minimizes the difference between the model distribution and the empirical distribution from a set of candidate feature functions.

Let Ω be the set of all packet classes, $\tilde{P}$ the empirical distribution of the training data over Ω, and $\mathcal{F}$ a set of candidate feature functions. The initial model distribution over Ω is $P_0(\omega) = \frac{1}{Z}$, $Z = |\Omega|$, which is a uniform distribution over Ω.

Now let P_i be a model with i feature functions selected

$$P_i(\omega) = \frac{1}{Z}\exp(\sum_{j=1}^{i} \lambda_j f_j(\omega)). \quad (4)$$

and we want to select the $i + 1^{st}$ feature function. Let g be a feature function in $\mathcal{F}\backslash\{f_1, \ldots f_i\}$ to be selected into the model and λ_g be its weight, then let

$$P_{i,\lambda_g,g}(\omega) = \frac{1}{Z'}\exp(\sum_i \lambda_i f_i(\omega))\exp(\lambda_g g), \quad (5)$$

and let

$$\begin{aligned} G_{P_i}(\lambda_g, g) &= D(\tilde{P}||P_i) - D(\tilde{P}||P_{i,\lambda_g,g}) \\ &= \lambda_g E_{\tilde{P}}(g) - \log E_{P_i}(\exp(\lambda_g g)), \end{aligned}$$

where $E_P(g)$ is the expected value of g with respect to the distribution of P. $G_{P_i}(\lambda_g, g)$ is a concave function with respect to λ_g, and

$$G_{P_i}(g) = \sup_{\lambda_g} G_{P_i}(\lambda_g, g) \tag{6}$$

is the maximum decrease of the K-L divergence that can be attained by adding g into the model. The feature function g with the largest gain $G_{P_i}(g)$ is selected as the $i+1^{st}$ feature function to the model.

In [11], it is also shown that for indicator candidate feature functions, there are closed form formulas related to the maxima of $G_{P_i}(\lambda_g, g)$, which makes it computationally easier. For more details on feature selection, please refer to [11] and [4].

4.2 Parameter estimation

After a new feature function is added to the log-linear model, the weights of all feature functions are updated. Given a set of training data and a set of selected feature functions $\{f_i\}$, the set of parameters is then estimated. Maximum Entropy estimation locates a set of parameters $\Lambda = \{\lambda_i\}$ in (3) for $\{f_i\}$ that minimizes the K-L divergence of $\tilde{P}$ with respect to P:

$$\Lambda = \arg\min_{\Lambda} \sum_{\omega \in \Omega} \tilde{P}(\omega) \log \frac{\tilde{P}(\omega)}{P(\omega)}. \tag{7}$$

There are a number of numerical methods that can be exploited. In our work, we use the L-BFGS Maximum Entropy estimation algorithm "tao_lmvm" implemented by Malouf in [8].

4.3 Model construction

Figure 1 shows the model construction algorithm. The model is built by iterating the above two steps until some stopping criterion is met. This stopping criterion can be either that the K-L divergence of P with respect to $\tilde{P}$ is less than some threshold value, or that the gain of adding a new feature function is too small to improve the model.

The feature functions are selected from a set of candidate feature functions. Since the domain Ω in our work consists of packet classes different in the protocols and the destination port numbers, our candidate feature function set comprises of three sets of indicator functions. The first set of indicator functions checks the packet's protocol information, the second set of indicator functions classify the packet's destination port number, and the third set checks both the packet's protocol information and the destination port number.

- Initial Data:

  A set of training data with empirical distribution $\tilde{P}$,
  a set of candidate feature functions $\mathcal{F}$,
  and an initial density model P_0, $P_0(\omega) = \frac{1}{Z}$, $Z = |\Omega|$

- Iterated steps:

```
(0) Set $n = 0$
(1) Feature selection
      For each feature function $g \in \mathcal{F}$, $g \notin \{f_i\}$, compute
         the gain $G_{P_n}(g)$
      Let $f_{n+1}$ be the feature function with the largest gain
(2) Parameter Estimation
      Update all the parameters and set $P_{n+1}$ to be the up-
         dated model
(3) Check the iteration stopping criterion
      If the iteration stopping criterion is not met, set $n =$
         $n + 1$, goto (1). Otherwise, return the learned
         model $P_{n+1}$.
```

Figure 1: Model construction algorithm

The training data used are pre-labeled by humans and the packets related to the labeled anomalies are not used in computing the empirical distribution $\tilde{P}$. In this way, we treat the packet classes distribution defined by the log-linear model in (3) from Maximum Entropy estimation as the baseline distribution, and are now able to compute the relative entropy of any given network traffic.

5 Detecting network traffic anomalies

The relative entropy shows the difference between the distribution of the packet classes in the current network traffic and the baseline distribution. If this difference is too large, it indicates that a portion of some packet classes that rarely appear in the training data increases significantly, or that appear regularly decreases significantly. In other words, this serves as an indication of the presence of an anomaly in the network traffic. Our current work only considers the anomalies where anomaly traffic increases.

We divide time into slots of fixed length δ. Suppose the traffic in a time slot contains the packet sequences $\{x_1, \ldots, x_n\}$, the empirical distribution $\tilde{P}$ of the packet classes in this time slot is

$$\tilde{P}(\omega) = \frac{\sum \mathbf{1}(x_i \in \omega)}{n}, \tag{8}$$

For each packet class, we define

$$D_{\tilde{P}\|P}(\omega) = \tilde{P}(\omega) \log \frac{\tilde{P}(\omega)}{P(\omega)}, \quad (9)$$

where P is the baseline distribution obtained from Maximum Entropy estimation. This produces a quantitative value that describes the distortion of the distribution for each packet class ω from that of the baseline distribution, and this is used as an indication of anomalies.

We then use a 'sliding window' detection approach. In each time slot, we record packet classes that have their divergences larger than a threshold d. If for a certain packet class ω, $D_{\tilde{P}\|P}(\omega) > d$ for more than h times in a window of W time slots, an alarm is raised together with the packet class information ω, which reveals the corresponding protocol and port number.

6 Experimental results

In this section, we present initial experimental results. The data are collected at the UMASS Internet gateway router using DAG cards made by Endace [3]. They consist of seven hours' traffic trace collected from 9:30am to 10:30am in the morning for a week from July 16th to July 22nd, 2004. All of these data are labeled by human inspection. In particular, we select a set of high volume flows, a set of nodes with high incoming or outgoing traffic, and a set of port numbers that have high volume of traffic. We then examine each of them to see whether there are anomalies. For more details of the trace collected, please refer to [4].

We use the data taken on July $20th$ as the training data set. The Maximum Entropy estimation algorithm is used to generate the baseline distribution of the packet classes from the training data. We set the stopping criterion for the construction algorithm to be whether the K-L difference of P with respect to $\tilde{P}$ is less than 0.01. By this criterion, the algorithm ended with a set of 362 feature functions.

As an example, we first show two cases of port scans that manifest themselves by increasing the $D_{\tilde{P}\|P}(\omega)$ value. The parameters used are set as $\delta = 1$ second, $d = 0.01$, $W = 60$ and $h = 30$. On July 19th, 2004, from 9:30am, when we began our data collection, to 9:37am, a host outside of the UMASS campus network performed a port scan at port 4899 by sending many SYN packets to different hosts in the UMASS campus network. Then from 9:46am to 9:51am, another host outside of the UMASS campus network performed another port scan at the same port. During these two time periods, the relative entropy of the packet class that represents SYN packets targeting at ports from 4824 to 4923 increased considerably, as shown in Figure 2. These two port scans were successfully detected by our relative entropy detection algorithm.

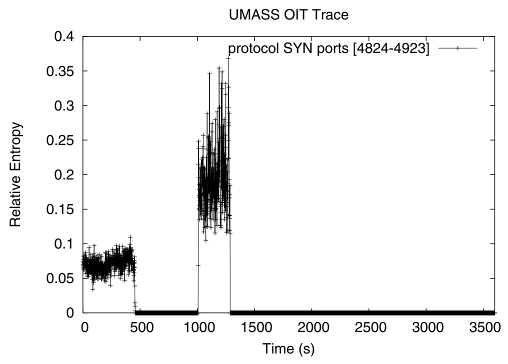


Figure 2: Relative entropy for packets of type SYN and destination port number from 4824 to 4923

We test the performance of the algorithm by running it over the remaining six human labeled data sets. The detection algorithm provides results at every time slot δ. If an anomaly is detected by the algorithm and there is a corresponding anomaly detected by human labeling, it is a *positive*. All anomalies detected by the algorithm corresponding to the same anomaly labeled by human are treated as a single positive. If there is no human labeled anomaly corresponding to the anomaly reported by the algorithm, it is called a *false positive*. Consecutive false positives are treated as a single false positive. Anomalies labeled by human but missed by the algorithm are called *false negatives*. In each case, the algorithm detects most of the anomalies located by human labeling. However, the algorithm also reports many 'false positives'. These 'false positives' are either 'flash crowds' phenomenons, high rate traffic that communicates with port numbers rarely seen in the training data, or traffic that we cannot tell what they are given the limited packet header information. For more details, please refer to [4].

In spite of the ambiguous situation concerning all the anomalies generated by the algorithm, we found that the experimental results regarding SYN packets give good results. Table 1 summarizes the algorithm performance in the experiments described above. The table also summarizes the performance of the algorithm in terms of precision, recall and F1. Let a be the number of positives, b the number of false positives, and c the number of false negatives, precision is defined as $a/(a+b)$, recall is defined as $a/(a+c)$ and F1 is defined as $2a/(2a+b+c)$. The table shows that the Maximum Entropy method detects most of the anomalies detected by human labeling with few false negatives and few false positives.

7 Implementation and practical issues

We are currently implementing the detection algorithm using an Intel IXP 1200 packet processing engine for

Date	Humanly labeled	Positive	False negative	False positive	Precision	Recall	F1
July 16	10	10	0	1	0.91	1	0.95
July 17	11	10	1	0	1	0.91	0.95
July 18	14	14	0	0	1	1	1
July 19	16	14	2	0	1	0.88	0.93
July 21	15	15	0	0	1	1	1
July 22	9	8	1	0	1	0.89	0.94

Table 1: Algorithm performance

routers [5], which has six processing engines, one control processor, and works at 200-MHz clock rate. The empirical distribution of the packet classes in the network traffic is read from the processing engine and compared to the baseline distribution every second. The baseline distribution is estimated offline. In practice, when the traffic is expected to experience certain changes, i.e. due to diurnal effects or planned network reconfiguration, the baseline distribution should be updated or retrained. How to do this is a topic of future research.

8 Conclusion

In this paper, we introduce our approach to detect anomalies in the network traffic using Maximum Entropy estimation and relative entropy. The packet distribution of the benign traffic is estimated using the Maximum Entropy framework and used as a baseline to detect the anomalies. The method is able to detect anomalies by inspecting only the current traffic instead of a change point detection approach. The experimental results show that it effectively detects anomalies in the network traffic including different kinds of SYN attacks and port scans. This anomaly detection method identifies the type of the anomaly detected and comes with low false positives. The method requires a constant memory and a computation time proportional to the traffic rate. Many interesting aspects of this approach still remain to be explored, and comparison with other methods such as Holt-Winter, when possible, will be useful.

Acknowledgements

We wish to thank Professor Paul Barford for useful comments and suggestions. Feedback from anonymous reviewers also helped to improve the work. This research is supported by NSF and DARPA under grants CNS-0085848 and F30602-00-2-0554. The data collection equipment was puchased under NSF grant EIA-0080119.

References

[1] BARFORD, P., KLINE, J., PLONKA, D., AND RON, A. A signal analysis of network traffic anomalies. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop* (2002).

[2] BRUTLAG, J. D. Aberrant behavior detection in time series for network service monitoring. In *Proceeding of the 14th Systems Administration Conference* (2000), pp. 139–146.

[3] ENDACE. http://www.endace.com.

[4] GU, Y., MCCALLUM, A., AND TOWSLEY, D. Detecting anomalies in network traffic using maximum entropy. Tech. rep., Department of Computer Science, UMASS, Amherst, 2005.

[5] INTEL CORP. Intel ixp 1200 network processor, 2000.

[6] INTERNET ASSIGNED NUMBERS AUTHORITY. http://www.iana.org/assignments/port-numbers.

[7] LEE, W., AND XIANG, D. Information-theoretic measures for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy* (2001), IEEE Computer Society, p. 130.

[8] MALOUF, R. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning* (2002).

[9] MCCALLUM, A. Efficiently inducing features of conditional random fields. In *Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03)* (2003).

[10] PAXSON, V. Bro: A system for detecting network intruders in real-time.

[11] PIETRA, S. D., PIETRA, V. D., AND LAFFERTY, J. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 4 (1997), 380–393.

[12] SNORT: THE OPEN SOURCE NETWORK INTRUSION DETECTION SYSTEM". http://www.snort.org/.

[13] STANIFORD, S., HOAGLAND, J., AND MCALERNEY, J. M. Practical automated detection of stealthy portscans. In *Proceedings of the IDS Workshop of the 7th Computer and Communications Security Conference* (2000).

[14] THOTTAN, M., AND JI, C. Anomaly detection in ip networks. *IEEE Trans. Signal Processing 51* (2003).

[15] WANG, H., ZHANG, D., AND SHIN, K. G. Detecting syn flooding attacks. In *Proceedings of IEEE INFOCOM* (2002).

Exploiting Underlying Structure for Detailed Reconstruction of an Internet-scale Event

Abhishek Kumar
Georgia Institute of Technology
akumar@cc.gatech.edu

Vern Paxson
ICSI
vern@icir.org

Nicholas Weaver
ICSI
nweaver@icsi.berkeley.edu

Abstract

Network "telescopes" that record packets sent to unused blocks of Internet address space have emerged as an important tool for observing Internet-scale events such as the spread of worms and the backscatter from flooding attacks that use spoofed source addresses. Current telescope analyses produce detailed tabulations of packet rates, victim population, and evolution over time. While such cataloging is a crucial first step in studying the telescope observations, incorporating an understanding of the underlying processes generating the observations allows us to construct detailed inferences about the broader "universe" in which the Internet-scale activity occurs, greatly enriching and deepening the analysis in the process.

In this work we apply such an analysis to the propagation of the *Witty* worm, a malicious and well-engineered worm that when released in March 2004 infected more than 12,000 hosts worldwide in 75 minutes. We show that by carefully exploiting the structure of the worm, especially its pseudo-random number generation, from limited and imperfect telescope data we can with high fidelity: extract the individual rate at which each infectee injected packets into the network *prior* to loss; correct distortions in the telescope data due to the worm's volume overwhelming the monitor; reveal the worm's inability to fully reach all of its potential victims; determine the number of disks attached to each infected machine; compute when each infectee was last booted, to sub-second accuracy; explore the "who infected whom" infection tree; uncover that the worm specifically targeted hosts at a US military base; and pinpoint *Patient Zero*, the initial point of infection, i.e., the IP address of the system the attacker used to unleash Witty.

1 Introduction

Network "telescopes"have recently emerged as important tools for observing Internet-scale events such as the spread of worms, the "backscatter" of responses from victims attacked by a flood of requests with spoofed source addresses, and incessant "background radiation" consisting of other anomalous traffic [10, 14, 15]. Telescopes record packets sent to unused blocks of Internet address space, with large ones using /8 blocks covering as much as 1/256 of the total address space. During network-wide anomalous events, such as the propagation of a worm, telescopes can collect a small yet significant slice of the worm's entire traffic. Previously, such logs of worm activity have been used to infer aggregate properties, such as the worm's infection rate (number of infected systems), the total scanning rate (number of worm copies sent per second), and the evolution of these quantities over time.

The fundamental premise of our work is that by carefully considering the underlying structure of the sources sending traffic to a telescope, we can extract a much more detailed reconstruction of such events. To this end, we analyze telescope observations of the *Witty* worm, a malicious and well-engineered[1] worm that spread worldwide in March 2004 in 75 minutes. We show that it is possible to reverse-engineer the state of each worm infectee's Pseudo-Random Number Generator (PRNG), which then allows us to recover the full set of actions undertaken by the worm. This process is greatly complicated by the worm's use of *periodic reseeding* of its PRNG, but we show it is possible to determine the new seeds, and in the process uncover detailed information about the individual hosts, including access bandwidth, up-time, and the number of physical drives attached. Our analysis also enables inferences about the network, such as shared bottlenecks and the presence or absence of losses on the path from infectees to the telescope. In addition, we uncover details unique to the propagation of the Witty worm: its failure to scan about 10% of the IP address space, the fact that it initially targeted a US military base, and the identity of *Patient Zero* — the host the worm's author used to release the worm.

Our analysis reveals systematic distortions in the data collected at telescopes and provides a means to correct this distortion, leading to more accurate estimates of quantities such as the worm's aggregate scan rate during its spread. It also identifies consequences of the specific topological placement of telescopes. In addition, detailed data about hitherto unmeasured quantities that emerges from our analysis holds promise to aid future worm simulations achieve a degree of realism well beyond today's abstract models. The techniques developed in our study, while specific to the Witty worm, highlight the power of such analysis, and provide a template for future analysis of similar events.

We organize the paper as follows. § 2 presents background material: the operation of network telescopes and related work, the functionality of Witty, and the structure of linear-congruential PRNGs. In § 3 we provide a roadmap to the subsequent analysis. We discuss how to reverse-engineer Witty's PRNG in § 4, and then use this to estimate access bandwidth and telescope measurement distortions in § 5. § 6 presents a technique for extracting the seeds used by individual infectees upon reseeding their PRNGs, enabling measurements of each infectee's system time and number of attached disks. This section also discusses our exploration of the possible infector-infectee relationships. We discuss broader consequences of our study in § 7 and conclude in § 8.

2 Background

Network Telescopes and Related Work. Network telescopes operate by monitoring unused or mostly-unused portions of the routed Internet address space, with the largest able to record traffic sent to /8 address blocks (16.7M addresses) [10, 22]. The telescope consists of a monitoring machine that passively records all packets headed to any of the addresses in the block. Since there are few or no actual machines using these addresses, traffic headed there is generally anomalous, and often malicious, in nature. Examples of traffic observed at network telescopes include port and address scans, "backscatter"from flooding attacks, misconfigurations, and the worm packets that are of immediate interest to this work.

The first major study performed using a network telescope was the analysis of backscatter by Moore et al. [14]. This study assessed the prevalence and characteristics of spoofed-source denial-of-service (DoS) attacks and the characteristics of the victim machines. The work built on the observation that most DoS tools that spoof source addresses pick addresses without a bias towards or against the telescope's observational range. The study also inferred victim behavior by noting that the response to spoofed packets will depend on the state of the victim, particularly whether there are services running on the targeted ports.

Telescopes have been the primary tool for understanding the Internet-wide spread of previous worms, beginning with Code Red [2, 20]. Since, for a random-scanning worm, the worm is as likely to contact a telescope address as a normal address, we can extrapolate from the telescope data to compute the worm's aggregate scanning rate as it spreads. In addition, from telescope data we can see which systems were infected, thus estimate the average worm scanning rate. For high-volume sources, we can also estimate a source's effective bandwidth based on the rate at which its packets arrive and adjusting for the telescope's "gatheringpower"(portion of entire space monitored).

A variation is the *distributed telescope*, which monitors a collection of disparate address ranges to create an overall picture [1, 4]. Although some phenomena [6, 2]) scan uniformly, others either have biases in their address selection [11, 12] or simply exclude some address ranges entirely [5, 16]. Using a distributed telescope allows more opportunity to observe nonuniform phenomenon, and also reveals that, even correcting for "local preference" biases present in some forms of randomized scanning, different telescopes observe quantitatively different phenomena [4].

The biggest limitation of telescopes is their passive nature, which often limits the information we can gather. One solution useful for some studies has been *active telescopes*: changing the telescope logic to either reply with SYN-ACKs to TCP SYNs in order to capture the resulting traffic [4], or implementing a more complex state machine [15] that emulates part of the protocol. These telescopes can disambiguate scans from different worms that target the same ports by observing subsequent transactions.

In this work we take a different approach for enhancing the results of telescope measurements: augmenting traces from a telescope with a detailed analysis of the structure of the sources sending the packets. One key insight is that the PRNG used to construct "random"addresses for a worm can leak the internal state of the PRNG. By combining the telescope data with our knowledge of the PRNG, we can then determine the internal state for each copy of the worm and see how this state evolves over time.

While there have been numerous studies of Internet worms, these have either focused on detailed analysis of the worm's exact workings, beginning with analysis of the 1988 Morris Worm [7, 19], or with aggregate propagation dynamics [23, 11, 18, 20, 13]. In contrast, our analysis aims to develop a detailed understanding of the individual infected hosts and how they interacted with the network.

Datasets. We used traces from two telescopes, operated by CAIDA [10] and the University of Wisconsin [22]. Both telescopes monitor /8 blocks of IP addresses. Since each /8 contains 1/256 of all valid IPv4 addresses, these telescopes see an equivalent fraction of scan traffic addressed to random destinations picked uniformly from the 32-bit IP address space. The CAIDA telescope logs every packet it receives, while the Wisconsin telescope samples the received packets at the rate of 1/10. The CAIDA trace [17] begins at 04:45 AM UTC, running for 75 minutes and totaling 45.5M packets. The Wisconsin trace runs from 04:45 AM UTC for 75 minutes, totaling 4.1M packets.

Functionality of the Witty worm. As chronicled by Shannon and Moore [18], an Internet worm was released on Friday March 19, 2004 at approximately 8:45 PM PST (4:45 AM UTC, March 20).

```
1.    Seed the PRNG using system time.
2.    Send 20,000 copies of self to random destinations.
3.    Open a physical disk chosen randomly between 0 & 7.
4.    If success:
5.        Overwrite a randomly chosen block.
6.        Goto line 1.
7.    Else:
8.        Goto line 2.
```

Figure 1: Functionality of the Witty worm

Its payload contained the phrase "`(^.^) insert witty message here (^.^)`" so it came to be known as the Witty worm. The worm targeted a buffer overflow vulnerability in several Internet Security Systems (ISS) network security products.

The vulnerability exploited was a stack-based overflow in the ICQ analyzer of these security products. When they received an ICQ packet, defined as any UDP packet with *source* port 4000 and the appropriate ICQ headers, they copied the packet into a fix ed-sized buffer on the stack in preparation for further analysis. The products executed this code path regardless of whether a server was listening for packets on the particular UDP destination port. In addition, some products could become infected while they *passively monitored* network links promiscuously, because they would attempt to analyze ICQ packets seen on the link even though they were not addressed to the local host.

Figure 1 shows a high-level description of the functionality of the Witty worm, as revealed by a disassembly [9]. The worm is quite compact, fitting in the first 675 bytes of a single UDP packet. Upon infecting a host, the worm first seeds its random number generator with the system time on the infected machine and then sends 20,000 copies of itself to random destinations. (These packets have a randomly selected *destination* port and a randomized amount of additional padding, but keep the source port fix ed.) After sending the 20,000 packets, the worm uses a three-bit random number to pick a disk via the `open` system call. If the call returns successfully, the worm overwrites a random block on the chosen disk, reseeds its PRNG, and goes back to sending 20,000 copies of itself. Otherwise, the worm jumps directly to the send loop, continuing for another 20,000 copies, *without* reseeding its PRNG.

The LC PRNG. The Witty worm used a simple feedback-based pseudo-random number generator (PRNG) of the form known as linear congruential (LC):

$$X_{i+1} = X_i * a + b \mod m \tag{1}$$

For a given m, picking effective values of a and b requires care lest the resulting sequences lack basic properties such as uniformity. One common parameterization is: $a = 214,013, b = 2,531,011, m = 2^{32}$.

With the above values of a, b, m, the LC PRNG generates a permutation of all the integers in $[0, m-1]$. A key point then is that with knowledge of any X_i, all subsequent pseudo-random numbers in the sequence can be generated by repeatedly applying Eqn 1. It is also possible to invert Eqn 1 to compute X_i if the value of X_{i+1} is known:

$$X_i = (X_{i+1} - b) * a^{-1} \mod m \tag{2}$$

where, for $a = 214,013$, $a^{-1} = 3,115,528,533$.

Eqns 1 and 2 provide us with the machinery to generate the entire sequence of random numbers as generated by an LC PRNG, either forwards or backwards, from any arbitrary starting point on the sequence. Thus, if we can extract *any* X_i, we can compute any other X_{i+n}, given n. However, it is important to note that most uses of pseudo-random numbers, including Witty's, do *not* directly expose any X_i, but rather extract a subset of X_i's bits and intermingle them with bits from additionally generated pseudo-random numbers, as detailed below.

3 Overview of our analysis

The first step in our analysis, covered in § 4, is to develop a way to uncover the state of an infectee's PRNG. It turns out that we can do so from the observation of just a single packet sent by the infectee and seen at the telescope. (Note, however, that if recovering the state required observing consecutive packets, we would likely often still be able to do so: while the telescopes record on average only one in 256 packets transmitted by an infectee, occasionally — i.e., roughly one time out of 256 — they will happen to record consecutive packets.)

An interesting fact revealed by careful inspection of the use of pseudo-random numbers by the Witty worm is that the worm does not manage to scan the entire 32-bit address space of the Internet, in spite of using a correct implementation of the PRNG. This analysis also reveals the identity of a special host that very likely was used to start the worm.

Once we have the crucial ability to determine the state of an infectee's PRNG, we can use this state to reproduce the worm's exact actions, which then allows us to compare the resulting generated packets with the actual packets seen at the telescope. This comparison yields a wealth of information about the host generating the packets and the network the packets traversed. First, we can determine the *access bandwidth* of the infectee, i.e., the capacity of the link to which its network interface connects. In addition, given this estimate we can explore significant flaws in the telescope observations, namely packet losses due to the finite bandwidth of the telescope's inbound link. These losses cause a systematic underestimation of infectee scan rates, but we design a mechanism to correct for this bias by calibrating against our measurements of the access bandwidth. We also highlight the impact of network location of telescopes on the observations they collect (§ 5).

We next observe that choosing a random disk (line 3 of Figure 1) consumes another pseudo-random number in ad-

```
rand(){
    # Note that 32-bit integers obviate the need for
    # a modulus operation here.
    $X = X * 214013 + 2531011$;
    return $X$; }
srand(seed){ $X = seed$; }
main(){
 1.     srand(get_tick_count());
 2.     for (i=0; i < 20,000; ++i)
 3.             dest_ip ← rand()$_{[0\cdots15]}$||rand()$_{[0\cdots15]}$;
 4.             dest_port ← rand()$_{[0\cdots15]}$;
 5.             packetsize ← 768+rand()$_{[0\cdots8]}$;
 6.             packetcontents ← top of stack;
 7.             sendto();
 8.     if(open(physicaldisk, rand()$_{[13\cdots15]}$))
 9.             overwrite_block(rand()$_{[0\cdots14]}$||0x4e20);
10.             goto 1;
11.     else goto 2; }
```

Figure 2: Pseudocode of the Witty worm

dition to those consumed by each transmitted packet. Observing such a discontinuity in the sequence of random numbers in packets from an infectee flags an attempted disk write and a potential reseeding of the infectee's PRNG. In § 6 we develop a detailed mechanism to detect the value of the seed at each such reseeding. As the seed at line 1 of Fig. 1 is set to the system time in msec since boot up, this mechanism allows us to estimate the boot time of individual infectees just by looking at the sequence of occasional packets received at the telescope. Once we know the PRNG's seed, we can precisely determine the random numbers it generates to synthesize the next 20,000 packets, and also the three-bit random number it uses next time to pick a physical disk to open. We can additionally deduce the success or failure of this `open` system call by whether the PRNG state for subsequent packets from the same infectee follow in the same series or not. Thus, this analysis reveals the number of physical disks on the infectee.

Lastly, knowledge of the seeds also provides access to the complete list of packets sent by the infectee. This allows us to infer infector-infectee relationships during the worm's propagation.

4 Analysis of Witty's PRNG

The first step in our analysis is to examine a disassembly of the binary code of the Witty worm [9]. Security researchers typically publish such disassemblies immediately after the release of a worm in an attempt to understand the worm's behavior and devise suitable countermeasures. Figure 2 shows the detailed pseudocode of the Witty worm as derived from one such disassembly [9]. The rand() function implements the Linear Congruential PRNG as discussed in § 2. In the rest of this section, we use the knowledge of the pseudocode to develop a technique for deducing the state of the PRNG at an infectee from any *single* packet sent by it. We also describe how as a consequence of the specific manner in which Witty uses the pseudo-random numbers, the worm fails to scan the entire IP address space, and also reveals the identity of *Patient Zero*.

Breaking the state of the PRNG at the infectee. The Witty worm constructs "random"destination IP addresses by concatenating the top 16 bits of two consecutive pseudo random numbers generated by its PRNG. In our notation, $X_{[0\cdots15]}$ represents the top 16 bits of the 32 bit number X, with bit 0 being the most significant. The destination port number is constructed by taking the top 16 bits of the next (third) random number. The packet size[2] itself is chosen by adding the top 9 bits of a fourth random number to 768. Thus, each packet sent by the Witty worm contains bits from four consecutive random numbers, corresponding to lines 3,4 and 5 in Fig. 2. If all 32 bits of any of these numbers were known, it would completely specify the state of the PRNG. But since only some of the bits from each of these numbers is known, we need to design a mechanism to retrieve all 32 bits of one of these numbers from the partial information contained in each packet.

To do so, if the first call to rand() returns X_i, then:

$$\begin{aligned} dest\_ip &= X_{i,[0\cdots15]}||X_{i+1,[0\cdots15]} \\ dest\_port &= X_{i+2,[0\cdots15]} \end{aligned}$$

where $||$ is the concatenation operation. Now, we know that X_i and X_{i+1} are related by Eqn 1, and so are X_{i+1} and X_{i+2}. Furthermore, there are only 65,536 (2^{16}) possibilities for the lower 16 bits of X_i, and only one of them is such that when used with $X_{i,[0\cdots15]}$ (available from the packet) the next two numbers generated by Eqn 1 have the same top 16 bits as $X_{i+1,[0\cdots15]}$ and $X_{i+2,[0\cdots15]}$, which are also observed in the received packet. In other words, there is only one 16-bit number Y that satisfies the following two equations simultaneously:

$$X_{i+1,[0\cdots15]} = (X_{i,[0\cdots15]}||Y * a \mod m)_{[0\cdots15]}$$

$$X_{i+2,[0\cdots15]} = ((X_{i,[0\cdots15]}||Y*a \mod m)*a \mod m)_{[0\cdots}$$

For each of the 2^{16} possible values of Y, verifying the first equality takes one addition and one multiplication.[3] Thus trying all 2^{16} possibilities is fairly inexpensive. For the small number of possible values of Y that satisfy the first equation, we try the second equation, and the value Y^* that satisfies both the equations gives us the lower sixteen bits of X_i (i.e., $X_{i,[16\cdots31]} = Y^*$). In our experiments, we found that on the average about two of the 2^{16} possible values satisfy the first equation, but there was always a unique value of Y^* that satisfied both the equations.

Why Witty fails to scan the entire address space. The first and somewhat surprising outcome from investigating how Witty constructs random destination addresses is the observation that Witty fails to scan the entire IP address space. This means that, while Witty spread at a very high speed (infecting 12,000 hosts in 75 minutes), due to a subtle error in its use of pseudo-random numbers about 10% of vulnerable hosts were never infected with the worm.

To understand this flaw in full detail, we first visit the motivation for the use of only the top 16 bits of the 32 bit results returned by Witty's LC PRNG. This was recommended by Knuth [8], who showed that the high order bits are "more random" than the lower order bits returned by the LC PRNG. Indeed, for this very reason, several implementations of the rand() function, including the default C library of Windows and SunOS, return a 15 bit number, even though their underlying LC PRNG uses the same parameters as the Witty worm and produces 32 bit numbers.

However, this advice was taken out of context by the author of the Witty worm. Knuth's advice applies when *uniform randomness* is the desired property, and is valid only when a small number of random bits are needed. For a worm trying to maximize the number of infected hosts, one reason for using random numbers while selecting destinations is to avoid detection by intrusion detection systems that readily detect sequential scans. A second reason is to maintain independence between the portions of the address-space scanned by individual infectees. Neither of these reasons actually requires the kind of "good randomness" provided by following Knuth's advice of picking only the higher order bits.

As discussed in § 2, for specific values of the parameters a, b and m, the LC PRNG is a *permutation* PRNG that generates a permutation of all integers in the range 0 to $m-1$. By the above definition, if the Witty worm were to use the entire 32 bits of a single output of its LC PRNG as a destination address, it would eventually generate each possible 32-bit number, hence successfully scanning the entire IP address space. (This would also of course make it trivial to recover the PRNG state.) However, the worm's author chose to use the concatenation of the top 16 bits of two consecutive random numbers from its PRNG. With this action, the guarantee that each possible 32-bit number will be generated is lost. In other words, there is no certainty that the set of 32-bit numbers generated in this manner will include all integers in the set $[0, 2^{32}-1]$.

We enumerated Witty's entire "orbit" and found that there are 431,554,560 32-bit numbers that can never be generated. This corresponds to 10.05% of the IP address space that was never scanned by Witty. On further investigation, we found these unscanned addresses to be fairly uniformly distributed over the 32-bit address space of IPv4. Hence, it is reasonable to assume that approximately the same fraction of the *populated* IP address space was missed by Witty. In other words, even though the portions of IP address space that are actually used (populated) are highly clustered, because the addresses that Witty misses are uniformly distributed over the space of 32-bit integers, it missed roughly the same fraction of address among the set of IP addresses in actual use.

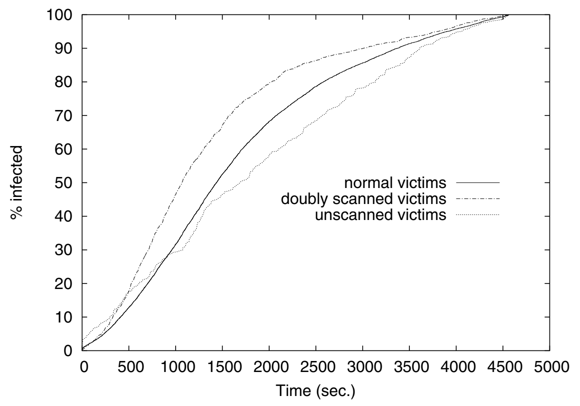


Figure 3: Growth curves for victims whose addresses were scanned once per orbit, twice per orbit, or not at all.

Observing that Witty does not visit some addresses at all, one might ask whether it visits some addresses more frequently than others. Stated more formally, given that the period of Witty's PRNG is 2^{32}, it must generate 2^{32} unique (X_i, X_{i+1}) pairs, from which it constructs 2^{32} 32-bit destination IP addresses. Since this set of 2^{32} addresses does not contain the 431,554,560 addresses missed by Witty, it must contain some repetitions. What is the nature of these repetitions? Interestingly, there are exactly 431,554,560 *other* 32-bit numbers that occur twice in this set, and no 32-bit numbers that occur three or more times. This is surprising because, in general, in lieu of the 431,554,560 missed numbers, one would expect some number to be visited twice, others to be visited thrice and so on. However, the peculiar structure of the sequence generated by the LC PRNG with specific parameter values created the situation that exactly the same number of other addresses were visited twice and none were visited more frequently.

During the first 75 minutes of the release of the Witty worm, the CAIDA telescope saw 12,451 unique IP addresses as infected. Following the above discussion, we classified these addresses into three classes. There were 10,638 (85.4%) addresses that were scanned just once in an orbit, i.e., addresses that experienced a normal scan rate. Another 1,409 addresses (11.3%) were scanned twice in an orbit, hence experiencing twice the normal growth rate. A third class of 404 (3.2%) addresses belonged to the set of addresses *never* scanned by the worm. At first blush one might wonder how these latter could possibly appear, but we can explain their presence as reflecting inclusion in an initial "hit list" (see below), operating in promiscuous mode, or aliasing due to multi-homing, NAT or DHCP.

Figure 3 compares the growth curves for the three classes of addresses. Notice how the worm spreads faster among the population of machines that experience double the normal scan rate. 1,000 sec from its release, Witty had infected half of the doubly-scanned addresses that it would infect in the first 75 min. On the other hand, in the normally-scanned population, it had only managed to infect about a third of the total victims that it would infect in 75 min. Later in the hour, the curve for the doubly-scanned addresses is flatter than that for the normally-scanned ones, indicating that most of the victims in the doubly-scanned population were already infected at that point.

The curve for infectees whose source address was *never* scanned by Witty is particularly interesting. Twelve of the never-scanned systems appear in the first 10 seconds of the worm's propagation, very strongly suggesting that they are part of an initial hit-list. This explains the early jump in the plot: it's not that such machines are overrepresented in the hit-list, rather they are underrepresented in the total infected population, making the hit-list propagation more significant for this population.

Another class of never-scanned infectees are those passively monitoring a network link. Because these operate in promiscuous mode, their "cross section" for becoming infected is magnified by the address range routed over the link. On average, these then will become infected much more rapidly than normal over even doubly-scanned hosts. We speculate that these infectees constitute the remainder of the early rise in the appearance of never-scanned systems. Later, the growth rate of the never-scanned systems substantially slows, lagging even the single-scanned addresses. Likely these remaining systems reflect infrequent aliasing due to multihoming, NAT, or DHCP.

Identifying Patient Zero. Along with "Can all addresses be reached by scans?", another question to ask is "Do all sources indeed travel on the PRNG orbit?" Surprisingly, the answer is No. There is a single Witty source that consistently fails to follow the orbit. Further inspection reveals that the source *(i)* always generates addresses of the form $A.B.A.B$ rather than $A.B.C.D$, *(ii)* does not randomize the packet size, and *(iii)* is present near the very beginning of the trace, but not before the worm itself begins propagating. That the source fails to follow the orbit clearly indicates that it is running *different* code than do all the others; that it does not appear prior to the worm's onset indicates that it is not a background scanner from earlier testing or probing (indeed, it sends valid Witty packets which could trigger an infection); and that it sends to sources of a limited form suggests a bug in its structure that went unnoticed due to a lack of testing of this particular Witty variant.

We argue that these peculiarities add up to a strong likelihood that this unique host reflects *Patient Zero*, the system used by the attacker to seed the worm initially. Patient Zero was not running the complete Witty worm but rather a (not fully tested) tool used to launch the worm. To our knowledge, this represents the first time that Patient Zero has been identified for a major worm outbreak.[4] We have conveyed the host's IP address (which corresponds to a European retail ISP) to law enforcement.

If all Patient Zero did was send packets of the form $A.B.A.B$ as we observed, then the worm would not have spread, as we detected no infectees with such addresses. However, as developed both above in discussing Figure 3 and later in § 6, the evidence is compelling that Patient Zero first worked through a "hit list" of known-vulnerable hosts before settling into its ineffective scanning pattern.

5 Bandwidth measurements

An important use of network telescopes lies in inferring the scanning rate of a worm by extrapolating from the observed packets rates from individual sources. In this section, we develop a technique based on our analysis of Witty's PRNG to estimate the access bandwidth of individual infectees. We then identify an obvious source of systematic error in extrapolation based techniques, namely the bottleneck at the telescope's inbound link, and suggest a solution to correct this error.

Estimating Infectee Access Bandwidth. The access bandwidth of the population of infected machines is an important variable in the dynamics of the spread of a worm. Using the ability to deduce the state of the PRNG at an infectee, we can infer this quantity, as follows. The Witty worm uses the `sendto` system call, which is a *blocking* system call by default in Windows: the call will not return till the packet has been successfully written to the buffer of the network interface. Thus, no worm packets are dropped either in the kernel or in the buffer of the network interface. But the network interface can clear out its buffer at most at its transmission speed. Thus, the use of blocking system calls indirectly clocks the rate of packet generation of the Witty worm to match the maximum transmission bandwidth of the network interface on the infectee.

We estimate the access bandwidth of an infectee as follows. Let P_i and P_j be two packets from the same infectee, received at the telescope at time t_i and t_j respectively. Using the mechanism developed in § 4 we can deduce X_i and X_j, the state of the PRNG at the sender when the two respective packets were sent. Now, we can simulate the LC PRNG with an initial state of X_i and repeatedly apply Eqn 1 till the state advances to X_j. The number of times Eqn 1 is applied to get from X_i to X_j is the value of $j - i$. Since it takes 4 cranks of the PRNG to construct each packet (lines 3–5, in Fig. 2), the total number of packets between P_i and P_j is $(j - i)/4$. Thus the access bandwidth of the infectee is approximately $average\_packetsize * (j-i)/4 * 1/(t_j - t_i)$. While we can compute it more precisely, since reproducing the PRNG sequence lets us extract the exact size of each intervening packet sent, for convenience we will often use the average payload size (1070 bytes including UDP, IP and Ethernet headers). Thus, the transmission rate can be computed as

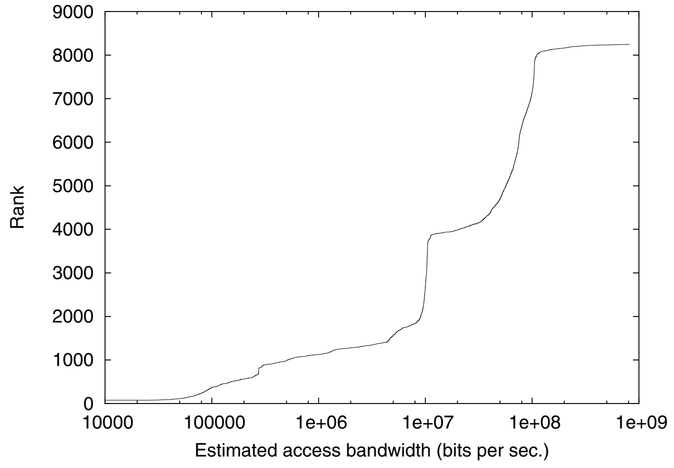


Figure 4: Access bandwidth of Witty infectees estimated using our technique.

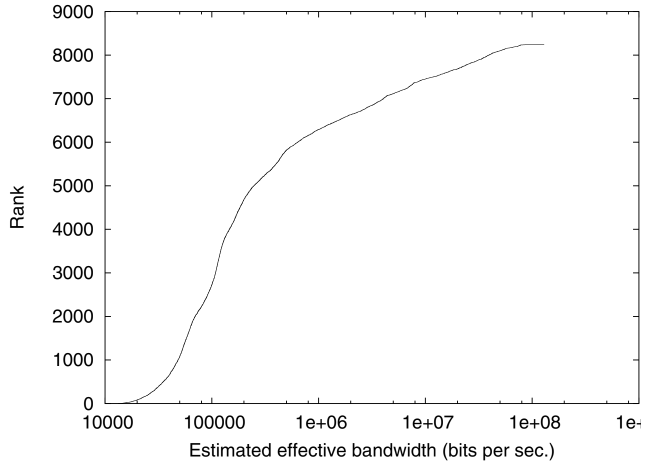


Figure 6: Effective bandwidth of Witty infectees.

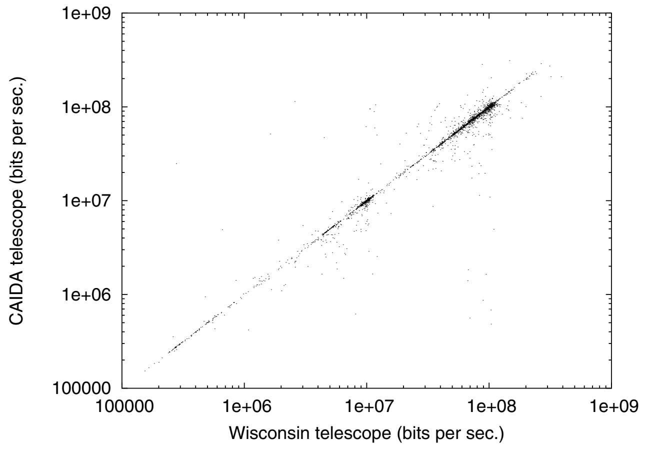


Figure 5: Comparison of estimated access bandwidth using data from two telescopes.

$\frac{(j-i)*1070*8}{4(t_j-t_i)} = 2140\frac{j-i}{t_j-t_i}$ bits per second.

Figure 4 shows the estimates of access bandwidth of infectees[5] that appeared at the CAIDA telescope from 05:01 AM to 06:01 AM UTC (i.e., starting about 15 min after the worm's release). The x-axis shows the estimated access bandwidth in bps on log scale, and the y-axis shows the rank of each infectee in increasing order. It is notable in the figure that about 25% of the infectees have an access bandwidth of 10 Mbps while about 50% have a bandwidth of 100 Mbps. This corresponds well with the popular workstation configurations connected to enterprise LANs (a likely description of a machine running the ISS software vulnerable to Witty), or to home machines that include an Ethernet segment connecting to a cable or DSL modem.

We use the second set of observations, collected independently at the Wisconsin telescope (located far from the CAIDA telescope), to test the accuracy of our estimation, as shown in Figure 5. Each point in the scatter plot represents a source observed in both datasets, with its x and y coordinates reflecting the estimates from the Wisconsin and CAIDA observations, respectively. Most points are located very close to the $y = x$ line, signifying close agreement. The small number of points (about 1%) that are significantly far from the $y = x$ line merit further investigation. We believe these reflect NAT effects invalidating our inferences concerning the amount of data a "single"source sends during a given interval.

Extrapolation-based estimation of effective bandwidth. Previous analyses of telescope data (e.g., [18]) used a simple extrapolation-based technique to estimate the bandwidth of the infectees. The reasoning is that given a telescope captures a /8 address block, it should see about 1/256 of the worm traffic. Thus, after computing the packets per second from individual infectees, one can extrapolate this observation by multiplying by 256 to estimate the total packets sent by the infectee in the corresponding period. Multiplying again by the average packet size (1070 bytes) gives the extrapolation-based estimate of the bandwidth of the infectee. Notice that this technique is not measuring the *access* bandwidth of the infectee, but rather the *effective* bandwidth, i.e., the rate at which packets from the infectee are actually delivered across the network.

Figure 6 shows the estimated bandwidth of the same population of infectees, computed using the extrapolation technique. The effective bandwidth so computed is significantly lower than the access bandwidth of the entire population. To explore this further, we draw a scatter-plot of the estimates using both techniques in Fig. 7. Each point corresponds to the PRNG-estimated access bandwidth (x axis) and extrapolation-based effective bandwidth (y axis). The modes at 10 and 100 Mbps in Fig. 4 manifest as clusters of points near the lines $x = 10^7$ and $x = 10^8$, re-

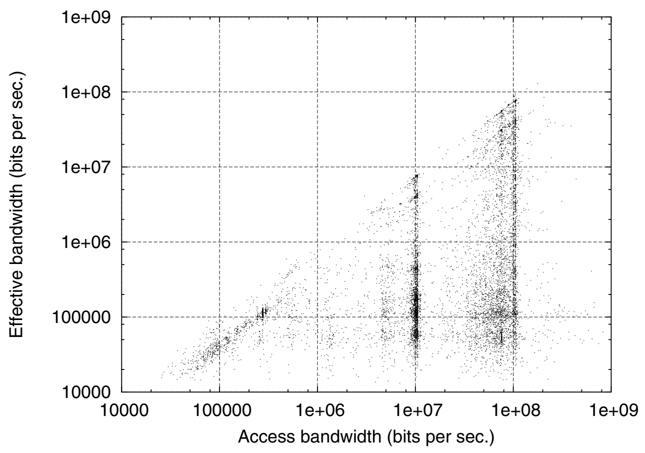


Figure 7: Scatter-plot of estimated bandwidth using the two techniques.

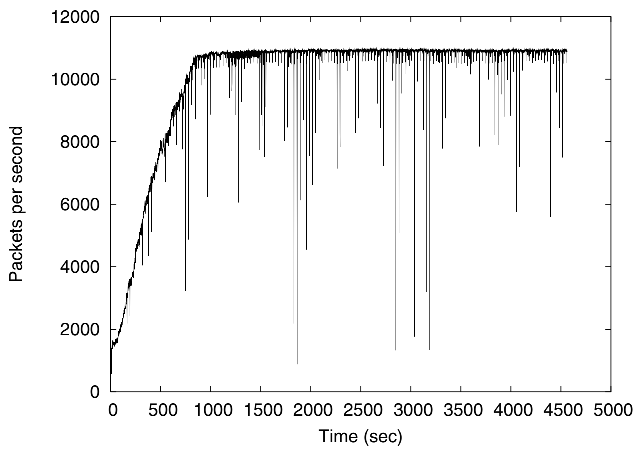


Figure 8: Aggregate worm traffic in pkts/sec as actually logged at the telescope.

spectively. As expected, all points lie below the diagonal, indicating that the effective bandwidth never exceeds the access bandwidth, and is often lower by a significant factor. During infections of bandwidth-limited worms, i.e., worms such as Witty that send fast enough to potentially consume all of the infectee's bandwidth, mild to severe congestion, engendering moderate to significant packet losses, is likely to occur in various portions of the network.

Another possible reason for observing diminished effective bandwidth is multiple infectees sharing a bottleneck, most likely because they reside within the same subnet and contend for a common uplink. Indeed, this effect is noticeable at /16 granularity. That is, sources exhibiting very high loss rates (effective bandwidth < 10% of access bandwidth) are significantly more likely to reside in /16 prefix es that include other infectees, than are sources with lower loss rates (effective > 50% access). For example, only 20% of the sources exhibiting high loss reside alone in their own /16, while 50% of those exhibiting lower loss do.

Telescope Fidelity. An important but easy-to-miss feature of Fig. 7 is that the upper envelope of the points is *not* the line $y = x$ but rather $y \approx 0.7x$, which shows up as the upper envelope of the scatter plot lying parallel to, but slightly below, the diagonal. This implies either a loss rate of nearly 30% for even the best connected infectees, or a systematic error in the observations. Further investigation immediately reveals the cause of the systematic error, namely congestion on the inbound link of the telescope. Figure 8 plots the packets received during one-second windows against time from the release of the worm. There is a clear ramp-up in aggregate packet rate during the initial 800 seconds after which it settles at approximately 11,000 pkts/sec. For an average packet size of 1,070 bytes, a rate of 11,000 pkts/sec corresponds to 95 Mbps, nearly the entire inbound bandwidth of 100 Mbps of the CAIDA

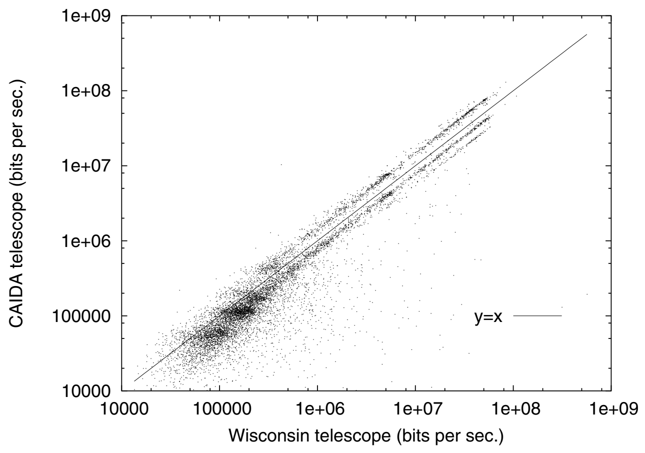


Figure 9: Comparison of effective bandwidth as estimated at the two telescopes.

telescope at that time.[6]

Fig. 8 suggests that the telescope may not have suffered any significant losses in the first 800 seconds of the spread of the worm. We verified this using a scatter-plot similar to Fig. 7, but only for data collected in the first 600 seconds of the infection. In that plot, omitted here due to lack of space, the upper envelope is indeed $y = x$, indicating that the best connected infectees were able to send packets unimpeded across the Internet, as fast as they could generate them.

A key point here is that our ability to determine access bandwidth allows us to *quantify* the 30% distortion[7] at the telescope due to its limited capacity. In the absence of this fine-grained analysis, we would have been limited to noting that the telescope saturated, but without knowing how much we were therefore missing.

Figure 9 shows a scatter-plot of the estimates of effective bandwidth as estimated from the observations at the

CAIDA $\geq$ Wisc.*1.05		Wisc. $\geq$CAIDA*1.05	
# Domains	TLD	# Domains	TLD
53	`.edu`	64	`.net`
17	`.net`	35	`.com`
7	`.jp`	9	`.edu`
5	`.nl`	7	`.cn`
5	`.com`	5	`.nl`
5	`.ca`	4	`.ru`
3	`.tw`	3	`.jp`
3	`.gov`	3	`.gov`
25	*other*	19	*other*

Table 1: Domains with divergent estimates of effective bandwidth.

two telescopes. We might expect these to agree, with most points lying close to the $y = x$ line, other than perhaps for differing losses due to saturation at the telescopes themselves, for which we can correct. Instead, we find two major clusters that lie approximately along $y = 1.4x$ and $y = x/1.2$. These lie parallel to the $y = x$ line due to the logscale on both axes. We see a smaller third cluster below the $y = x$ line, too. These clusters indicate systematic divergence in the telescope observations, and *not* simply a case of one telescope suffering more saturation losses than the other, which would result in a *single* line either above or below $y = x$.

To analyze this effect, we took all of the sources with an effective bandwidth estimate from both telescopes of more than 10 Mbps. We resolved each of these to domain names via reverse DNS lookups, taking the domain of the responding nameserver if no PTR record existed. We then selected a representative for each of the unique second-level domains present among these, totaling 900. Of these, only 29 domains had estimates at the two telescopes that agreed within 5% after correcting for systematic telescope loss. For 423 domains, the corrected estimates at CAIDA exceeded those at Wisconsin by 5% or more, while the remaining 448 had estimates at Wisconsin that exceeded CAIDA's by 5% or more.

Table 1 lists the top-level domains for the unique second-level domains that demonstrated $\geq$ 5% divergence in estimated effective bandwidth. Owing to its connection to Internet-2, the CAIDA telescope saw packets from `.edu` with significantly fewer losses than the Wisconsin telescope, which in turn had a better reachability from hosts in the `.net` and `.com` domains. Clearly, telescopes are not "ideal"devices, with perfectly balanced connectivity to the rest of the Internet, as implicitly assumed by extrapolation-based techniques. Rather, what a telescope sees during an event of large enough volume to saturate high-capacity Internet links is dictated by its specific location on the Internet topology. This finding complements that of [4], which found that the (low-volume) background radiation seen at different telescopes likewise varies significantly with location, beyond just the bias of some malware to prefer nearby addresses when scanning.

6 Deducing the seed

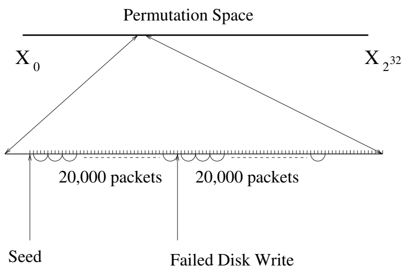
Cracking the seeds — System uptime. We now describe how we can use the telescope observations to deduce the exact values of the seeds used to (re)initialize Witty's PRNG. Recall from Fig. 2 that the Witty worm attempts to open a disk after every 20,000 packets, and reseeds its PRNG on success. To get a seed with reasonable local entropy, Witty uses the value returned by the `Get_Tick_Count` system call, a counter set to zero at boot time and incremented every millisecond.

In § 4 we have developed the capability to reverse-engineer the state of the PRNG at an infectee from packets received at the telescope. Additionally, Eqns 1 and 2 give us the ability to crank the PRNG forwards and backwards to determine the state at preceding and successive packets. Now, for a packet received at the telescope, if we could identify the precise number of calls to the function `rand` between the reseeding of the PRNG and the generation of the packet, simply cranking the PRNG backwards the same number of steps would reveal the value of the seed. The difficulty here is that for a given packet we do *not* know which "generation"it is since the PRNG was seeded. (Recall that we only see a few of every thousand packets sent.) We thus have to resort to a more circuitous technique.

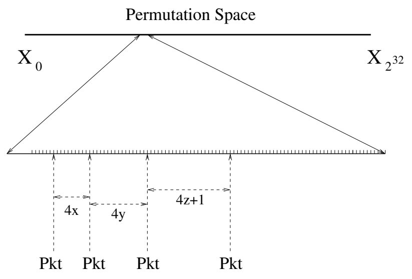
We split the description of our approach into two parts: a technique for identifying a small range in the orbit (permutation sequence) of the PRNG where the seed must lie, and a geometric algorithm for finding the seeds from this candidate set.

Identifying a limited range within which the seed must lie. Figure 10 shows a graphical view of our technique for restricting the range where the seed can potentially lie. Figure 10(a) shows the sequence of packets as generated at the infectee. The straight line at the top of the figure represents the permutation-space of the PRNG, i.e., the sequence of numbers $X_0, X_1, \cdots, X_{2^{32}-1}$ as generated by the PRNG. The second horizontal line in the middle of the figure represents a small section of this sequence, blown-up to show the individual numbers in the sequence as ticks on the horizontal line. Notice how each packet consumes exactly four random numbers, represented by the small arcs straddling four ticks.

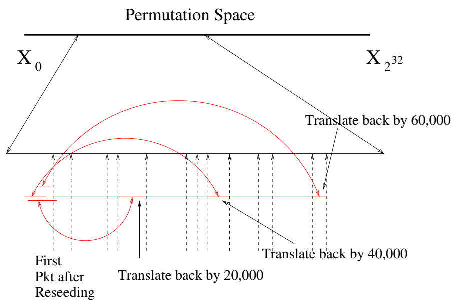
Only a small fraction of packets generated at the infectee reach the telescope. Figure 10(b) shows four such packets. By cranking forward from the PRNG's state at the first packet until the PRNG reaches the state at the second packet, we can determine the precise number of calls to the `rand` function in the intervening period. In other words, if we start from the state corresponding to the first packet and apply Eqn 1 repeatedly, we will eventually (though see



(a) Sequence of packets generated at the infectee.



(b) Packets seen at the telescope. Notice how packets immediately before or after a failed disk-write are separated by $4z + 1$ cranks of the PRNG rather than $4z$.



(c) Translating these *special* intervals back by multiples of 20,000 gives bounds on where the seed can lie.

Figure 10: Restricting the range where potential seeds can lie.

below) reach the state corresponding to the second packet, and counting the number of times Eqn 1 was applied gives us the precise number of random numbers generated between the departure of these two packets from the infectee. Note that since each packet consumes four random numbers (the inner loop of lines 2–7 in Fig. 2), the number of random numbers will be a multiple of four.

However, sometimes we find the state for a packet received at the telescope does *not* lie within a reasonable number of steps (300,000 calls to the PRNG) from the state of the preceding packet from the same infectee. This signifies a potential reseeding event: the worm finished its batch of 20,000 packets and attempted to open a disk to overwrite a random block. Recall that there are two possibilities: the random disk picked by the worm exists, in which case it overwrites a random block and (regardless of the success of that attempted overwrite) reseeds the PRNG, jumping to an arbitrary location in the permutation space (control flowing through lines 8→9→10→1→2 in Fig. 2); or the disk does not exist, in which case the worm continues for another 20,000 packets *without* reseeding (control flowing through lines 8→11→2 in Fig. 2). Note that *in either case* the worm consumes a random number in picking the disk.

Thus, every time the worm finishes a batch of 20,000 packets, we will see a discontinuity in the usual pattern of $4z$ random numbers between observed packets. We will instead either find that the packets correspond to $4z + 1$ random numbers between them (disk open failed, no reseeding); or that they have no discernible correspondence (disk open succeeded, PRNG reseeded and now generating from a different point in the permutation space).

This gives us the ability to identify intervals within which either failed disk writes occurred, or reseeding events occurred. Consider the interval straddled by the first failed disk write after a successful reseeding. Since the worm attempts disk writes every 20,000 packets, this interval translated back by 20,000 packets (80,000 calls to the PRNG) must straddle the seed. In other words, the beginning of this special interval must lie no more than 20,000 packets away from the reseeding event, and its end must lie no less than that distance away. This gives us upper *and* lower bounds on where the reseeding must have occurred. A key point is that these bounds are *in addition* to the bounds we obtain from observing that the worm reseeded. Similarly, if the worm fails at its next disk write attempt too, the interval straddling *that* failed write, when translated backwards by 40,000 packets (160,000 calls to the PRNG), gives us another pair of lower and upper bounds on where the seed must lie. Continuing this chain of reasoning, we can find multiple upper and lower bounds. We then take the *max* of all lower bounds and the *min* of all upper bounds to get the tightest bounds, per Figure 10(c).

A geometric algorithm to detect the seeds. Given this procedure, for each reseeding event we can find a limited range of potential in the permutation space wherein the new seed must lie. (I.e., the possible seeds are consecutive over a range in the permutation space of the consecutive 32-bit random numbers as produced by the LC PRNG; they are *not* consecutive 32-bit integers.) Note, however, that this may still include hundreds or thousands of candidates, scattered over the full range of 32-bit integers.

Which is the correct one? We proceed by leveraging two key points: *(i)* for most sources we can find numerous reseeding events, and *(ii)* the actual seeds at each event are strongly related to one another by the *amount of time* that elapsed between the events, since the seeds are *clock readings*. Regarding this second point, recall that the seeds are read off a counter that tracks the number of milliseconds since system boot-up. Clearly, this value increases linearly with time. So if we observe two reseeding events with timestamps (at the telescope) of t_1 and t_2, with corresponding seeds S_1 and S_2, then because clocks progress linearly with time, $(S_2 - S_1) \approx (t_2 - t_1)$. In other words, if the infectee reseeded twice, then the value of the seeds

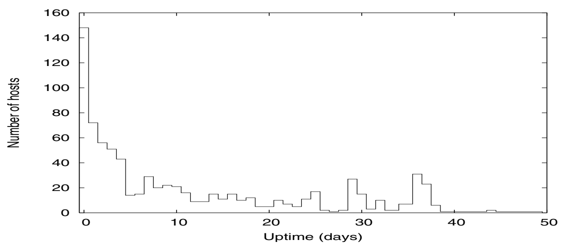


Figure 11: Number of infectees with a system uptime of the given number of days.

must differ by approximately the same amount as the difference in milliseconds in the timestamps of the two packets seen immediately after these reseedings at the telescope. Extending this reasoning to k reseeding events, we get $(S_j - S_i) \approx (t_j - t_i), \forall i, j : 1 \leq i, j \leq k$. This implies that the k points (t_i, S_i) should (approximately) lie along a straight line with slope 1 (angle of $45°$) when plotting potential seed value against time.

We now describe a geometric algorithm to detect such a set of points in a 2-dimensional plane. The key observation is that when k points lie close to a straight line of a given slope, then looking from any one of these points along that slope, the remaining points should appear clustered in a very narrow band. More formally, if we project an angular beam of width δ from any one of these points, then the remaining points should lie within the beam, for reasonably small values of δ. On the other hand, other, randomly scattered points on the plane will see a very small number of other points in the beam projected from them.

The algorithm follows directly from this observation. It proceeds in iterations. Within an iteration, we project a beam of width $\delta = \arctan 0.1 \approx 0.1$ along the $45°$ line from each point in the plane. The point is assigned a score equal to the number of other points that lie in its beam. Actual seeds are likely to get a high score because they would all lie roughly along a $45°$ line. At the end of the iteration, all points with a score smaller than some threshold (say $k/2$) are discarded. Repeating this process in subsequent iterations quickly eliminates all but the k seeds, which keep supporting high scores for each other in all iterations.

We find this algorithm highly effective given enough reseeding events. Figure 11 presents the results of the computation of system uptime of 784 machines in the infectee population. These infectees were chosen from the set that contributed enough packets to allow us to use our mechanism for estimating the seed. Since the counter used by Witty to reseed its PRNG is only 32 bits wide, it will wraparound every 2^{32} milliseconds, which is approximately 49.7 days. The results could potentially be distorted due to this effect (but see below).

There is a clear domination of short-lived machines, with approximately 47% having uptimes of less than fi ve days. On the other hand, there are just fi ve machines that had an uptime of more than 40 days. The sharp drop-off above 40 days leads us to conclude that the effects due to the wrapping-around of the counter are negligible.

The highest number of machines were booted on the same day as the spread of the worm. There are prominent troughs during the weekends — recall that the worm was released on a Friday evening Pacific Time, so the nearest weekend had passed 5 days previously — and heightened activity during the working days.

One feature that stands out is the presence of two modes, one at 29 days and the second at 36/37 days. On further investigation, we found that the machines in the first mode all belonged to a set of 135 infectees from the same /16 address block, and traceroutes revealed they were situated at a single US military installation. Similarly, machines in the second mode belonged to a group of 81 infectees from another /16 address block, belonging to an educational institution. However, while machines in the second group appeared at the telescope one-by-one throughout the infection period, 110 of the 135 machines in the first group appeared at the telescope within 10 seconds of Witty's onset. Since such a fast spread is not feasible by random scanning of the address space, the authors of [18] concluded that these machines were either part of a hit-list or were already compromised and under the control of the attacker. Because we can fit the actions of these infectees with running the full Witty code, including PRNG reseeding patterns that match the process of overwriting disk blocks, this provides evidence that these machines were not specially controlled by the attacker (unlike the *Patient Zero* machine), and thus we conclude that they likely constitute a hit-list. (We investigated an alternate explanation that instead these machines were passively monitoring large address regions and hence were infected much more quickly, but can discount this possibility because a "lineage"analysis reveals that a significant number of the machines did not receive any infection packets on even their entire local /16 prior to their own scanning activity arriving at the telescope. Additionally, these systems' IP addresses also suggest local monitors, rather than a collection of global monitors on a large address space.) Returning then to the fact that these machines were all rebooted exactly 29 days before the onset of the worm, we speculate that the reboot was due to a facility-wide system upgrade; perhaps the installation of system software such as Microsoft updates (a critical update had been released on Feb. 10, about 10 days before the simultaneous system reboots), or perhaps the installation of the vulnerable ISS products themselves. We might then speculate that the attacker *knew* about the ISS installation at the site (thus enabling them to construct a hit-list), which, along with the attacker's rapid construction of the worm indicating they likely knew about the vulnerability in advance [21], suggests that the attacker was an ISS "insider"

Number of disks. Once we can recover the seed used at

Number of Disks	1	2	3	4	5	6	7
Number of Infectees	52	32	12	2	2	0	0

Table 2: Disk counts of 100 infectees.

the beginning of a sequence of packets, we can use its value as an anchor to mark off the precise subsequent actions of the worm. Recall from Fig. 2 that the worm generates exactly 20,000 packets in its inner loop, using 80,000 random numbers in the process. After exiting the inner loop, the worm uses three bits from the next random number to decide which physical disk it will attempt to open. Starting from the seed, this is exactly the 80,001th number in the sequence generated by the PRNG. Thus, knowledge of the seed tells us exactly which disk the worm attempts to open. Furthermore, as discussed above we can tell whether this attempt succeeded based on whether the worm reseeds after the attempt. We can therefore estimate the number of disks on the infectee, based on which of the attempts for drives in the range 0 to 7 lead to a successful return from the `open` system call. Table 2 shows the number of disks for 100 infectees, calculated using this approach. The majority of infectees had just one or two disks, while we find a few with up to fi ve disks. Since the installation of end-system fire wall software was a prerequisite for infection by Witty, the infectee population is more likely to contain production servers with multiple disks.

Exploration of infection graph. Knowledge of the precise seeds allows us to reconstruct the complete list of packets sent by each infectee. Additionally, the large size of our telescope allows us to detect an infectee within the first few seconds (few hundred packets) of its infection. Therefore if an infectee is first seen at a time T, we can inspect the list of packets sent by all other infectees active within a short preceding interval, say $(T - 10 \text{ sec}, T)$, to see which sent a packet to the new infectee, and thus is the infectee's likely "infector" to select the most likely "infector".

The probability of more than one infectee sending a worm packet to the same new infectee at the time of its infection is quite low. With about 11,000 pkts/sec seen at a telescope with 1/256 of the entire Internet address space, and suffering 30% losses due to congestion (§ 5), the aggregate scanning rate of the worm comes out to around $256 \cdot 11,000/0.7 \approx 4 \cdot 10^6$ pkts/sec. With more than $4 \cdot 10^9$ addresses to scan, the probability that more than one infectee scans the same address within the same 10 second interval is around 1%.

Figure 12 shows scan packets from infected sources that targeted other infectees seen at the telescope. The x-coordinate gives t_{scan}, the packet's estimated sending time, and the y-coordinate gives the difference between $t_{\text{infection}}$, the time when the target infectee first appeared at the telescope, and t_{scan}. A small positive value of $t_{\text{infection}} - t_{\text{scan}}$ raises strong suspicions that the given scan packet is responsible for infecting the given target. Negative values mean the target was already infected, while larger positive values imply the scan failed to infect the target for some reason — it was lost,[8] or blocked due to the random destination port it used, or simply the target was not connected to the Internet at that time. (Note that the asymptotic curves at the top and bottom correspond to truncation effects reflecting the upper and lower bounds on infection times.)

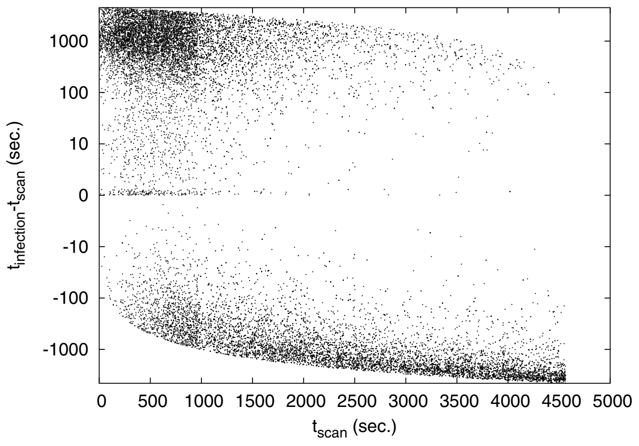


Figure 12: Scans from infectees, targeted to other victims.

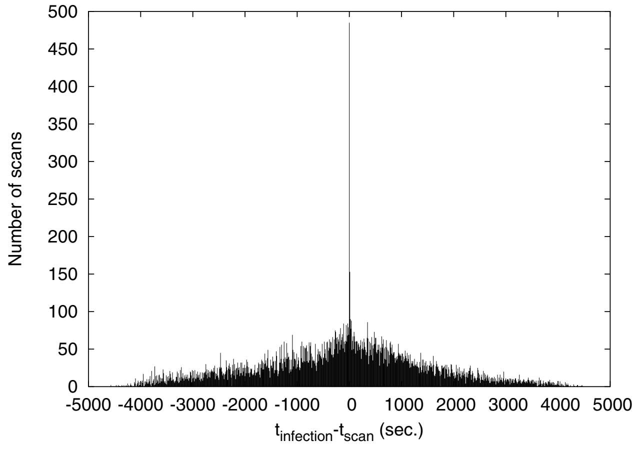


Figure 13: Number of scans in 10 second buckets.

The clusters at extreme values of $t_{\text{infection}} - t_{\text{scan}}$ in Figure 12 mask a very sharp additional cluster, even using the log-scaling. This lies in the region $0 < t_{\text{infection}} - t_{\text{scan}} \leq 10$. In Figure 13, we plot the number of scans in 10 second buckets against $t_{\text{infection}} - t_{\text{scan}}$. The very central sharp peak corresponds to the interval 0-to-10 seconds — a clear mark of the dispatch of a successful scan closely followed by the appearance of the victim at the telescope. We plan to continue our investigation of infector-infectee relationships, hoping to produce an extensive "lineage" of infection chains for use in models of worm propagation.

7 Discussion

While we have focused on the Witty worm in this paper, the key idea is much broader. Our analysis demonstrates the potential richness of information embedded in network telescope observations, ready to be revealed if we can frame a precise model of the underlying processes generating the observations. Here we discuss the breadth and limitations of our analysis, and examine general insights beyond the specific instance of the Witty worm.

Candidates for similar analysis. The binary code of all Internet worms is available by definition, making them candidates for disassembly and analysis. Similarly, copies of many scanning and flooding tools have been captured by white hat researchers, and traces observed at telescopes of probing or attack traffic (or backscatter) from the operation of such tools provide candidates for similar analysis. A preliminary assessment we performed of ten well-known DoS attack tools revealed that six of them use simple PRNGs with unsophisticated seeds, while the other four use no random number generation at all. Even with limited knowledge of the operation of such tools, we should in principle be able to analyze logs of their attack traffic or backscatter with a similar intent of reconstructing the sequence of events in the automation of the attack, potentially leading to information about the attacking hosts, their interaction with the network, and other forensic clues.

Diversity of PRNGs. Our analysis was greatly facilitated by the use of a linear congruential PRNG by Witty's author. Reverse-engineering the state of a more complex PRNG could be much more difficult. In the extreme, a worm using a cryptographically strong hash function with a well-chosen key as its PRNG would greatly resist such reverse engineering. However, there are several practical reasons that support the likelihood of many attackers using simpler PRNGs.

Implementing good PRNGs is a complicated task [8], especially when constrained by limits on code size and the difficulty of incorporating linkable libraries. Large-scale worms benefit greatly from as self-contained a design as possible, with few dependencies on platform support, to maximize the set of potential victims. Worms have also proven difficult to fully debug — virtually all large-scale worms have exhibited significant bugs — which likewise argues for keeping components as simple as possible. Historically, worm authors have struggled to implement even the LC PRNG correctly. The initial version of Code Red failed to seed the PRNG with any entropy, leading to all copies of the worm scanning exactly the same sequence of addresses [2]. Slammer's PRNG implementation had three serious errors, one where the author used a value of the parameter b in the LC equation (Eqn. 1) that was larger than the correct value by 1 due to an incorrect 2's complement conversion, another where this value was subtracted from instead of added to the term aX_i in Eqn 1, and finally the (mis)use of an **OR** instruction rather than **XOR** to clear a key register [11]. In addition, sources of local entropy at hosts are often limited to a few system variables, complicating the task of seeding the PRNG in a fashion strong enough to resist analysis. Thus it is conceivable that worm authors will have difficulty implementing bug-free, compact versions of sophisticated PRNGs.

In addition, today's worm authors have little incentive to implement a complex PRNG. As long as their goals are confined to effectively scanning the IP address space and maximizing the worm's infection rate, simple PRNGs suffice. Hiding one's tracks while releasing a worm can already be accomplished by using a chain of compromised victims as stepping stones. Indeed, the fact that Witty's author left *Patient Zero* running with a separate program for spreading the worm was purely a mistake on his/her part. As discussed earlier, the code it ran scanned a very small subset of the IP address space, and did not manage to produce even one infection during scanning.

Thus, there are significant factors that may lead to the continued use by worms of simple PRNGs such as LC, which, along with the availability of disassembled code, will facilitate the development of structural models of worm behavior to use in conjunction with telescope observations for detailed reconstructions.

General observations from this work. Our study has leveraged the special conditions produced by a worm's release to measure numerous features of its victim population and the network over which it spread. While specific estimation tricks developed in this paper might not apply to other telescope observations in a "cookbook"manner, the insight that telescope observations carry rich information that can be heavily mined armed with a sufficiently detailed model of the underlying source processes is of major significance for the future study of such data.

Understanding the structure of the scanning techniques used by worms (and empirical data on hitherto unmeasured quantities such as distribution of access bandwidth) can be crucial for developing correct models of their spread — a case made for example by our observation of the doubly-scanned and never-scanned portions of the address space, and their multi-factored impact on the worm's growth.

Finally, we would emphasize that the extraction of the features we have assessed was a labor-intensive process. Indeed, for many of them we did not initially apprehend even the possibility of analyzing them. This highlights not only the difficulty of such a forensic undertaking, but also its serendipitous nature. The latter holds promise that observations of other Internet-scale events in the future, even those of significantly different details or nature, will likely remain open to the possibility of such analysis.

8 Conclusions

A worm's propagation is a rare but spectacular event in today's networks. Apart from the obvious disruptions and damage, worms also stress the network in unique ways and at scales unmatched by any controlled measurement experiments. One could say that a worm's release illuminates, for a few moments, dark corners of the network just as supernovae illuminate dark and distant corners of the universe, providing rich observations to telescopes that gather a mere sliver of the enormous radiant flux. But within the overwhelming mass of observed data lies a very structured process that can be deciphered and understood — if studied with the correct model.

We have shown how a fine-grained understanding of the exact control flow of a particular worm — especially its seeding and use of a pseudo-random number generator — when coupled with network telescope data enables a detailed reconstruction of nearly the entire chain of events that followed the worm's release. In the process we have unearthed measurements of quantities such as access bandwidth and system up-time that are otherwise unobservable to the "naked eye" of researchers studying systems from afar. These measurements have applicability to a number of modeling and simulation studies, both in particular to worm propagation analysis, and more generally as a source of rarely-available empirical data. Finally, we have demonstrated the forensic power that such analysis can provide, marshalling strong evidence that the Witty worm specifically targeted a US military base and was launched via an IP address corresponding to a European ISP.

Acknowledgments

This work was supported by the National Science Foundation under the following grants: Collaborative Cybertrust NSF-0433702, ITR/ANI-0205519, NRT-0335290, and ANI-0238315, for which we are grateful. We thank Colleen Shannon and David Moore at CAIDA, and Paul Barford and Vinod Yegneswaran at the University of Wisconsin for providing access to the telescope traces and answering numerous questions about them, and our CCIED colleagues and Ellen Zegura for valuable feedback.

Support for the Witty Worm Dataset and the UCSD Network Telescope are provided by Cisco Systems, Limelight Networks, the US Dept. Homeland Security, the National Science Foundation, and CAIDA, DARPA, Digital Envoy, and CAIDA Members.

References

[1] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson. The Internet motion sensor: A distributed blackhole monitoring system. In *Proc. NDSS*, 2005.

[2] CAIDA. CAIDA Analysis of Code-Red, http://www.caida.org/analysis/security/code-red/.

[3] CERT. CERT Advisory CA-1999-04 Melissa Macro Virus, http://www.cert.org/advisories/CA-1999-04.html.

[4] Evan Cooke, Michael Bailey, Z. Morley Mao, David Watson, Farnam Jahanian, and Danny McPherson. Toward understanding distributed blackhole placement. In *Proc. ACM CCS Workshop on Rapid Malcode (WORM)*, October 2004.

[5] Domas Mituzas. FreeBSD Scalper Worm, http://www.dammit.lt/apache-worm/.

[6] eEye Digital Security. .ida "Code Red" Worm, http://www.eeye.com/html/Research/Advisories/AL20010717.html.

[7] Mark Eichin and Jon Rochlis. With microscope and tweezers: An analysis of the Internet virus of november 1988. In *Proc. IEEE Symposium on Research in Security and Privacy*, 1989.

[8] Donald E. Knuth. *The Art of Computer Programming, Second Edition*, volume 2, Seminumerical Algorithms. Addison-Wesley, 1981.

[9] K. Kortchinsky. Black Ice worm disassembly. http://www.caida.org/analysis/security/witty/BlackIceWorm.html.

[10] D. Moore, C. Shannon, G. Voelker, and S. Savage. Network telescopes: Technical report. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), July 2004.

[11] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer Worm. *IEEE Security & Privacy*, pages 33–39, July/August 2003.

[12] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The Spread of the Sapphire/Slammer Worm, 2003.

[13] David Moore, Colleen Shannon, and k claffy. Code-Red: a Case Study on the Spread and Victims of an Internet Worm. In *Proceedings of the Second Internet Measurement Workshop*, pages 273–284, November 2002.

[14] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet Denial-of-Service Activity. In *Proceedings of the 10th USENIX Security Symposium*, pages 9–22. USENIX, August 2001.

[15] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. Characteristics of Internet background radiation. In *Proc. ACM Internet Measurement Conference*, October 2004.

[16] F secure Inc. Global slapper worm information center, http://www.f-secure.com/slapper/.

[17] C. Shannon and D. Moore. The caida dataset on the witty worm, March 19-24 2004. http://www.caida.org/passive/witty/.

[18] C. Shannon and D. Moore. The spread of the Witty worm. *IEEE Security and Privacy*, 2(4):46–50, August 2004.

[19] Eugene Spafford. The Internet worm program: An analysis. purdue technical report csd-tr-823, 1988.

[20] Stuart Staniford and Vern Paxson and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, August 2002.

[21] Nicholas Weaver and Dan Ellis. Reflections on Witty: Analyzing the attacker. *;login:*, pages 34–37, June 2004.

[22] V. Yegneswaran, P. Barford, and D. Plonka. On the design and utility of Internet sinks for network abuse monitoring. In *Proc. of Symposium on Recent Advances in Intrusion Detection*, September 2004.

[23] Cliff Changchun Zou, Weibo Gong, and Don Towsley. Code Red Worm Propagation Modeling and Analysis. In *Proceedings of the ACM CCS 2002 conference*, November 2002.

Notes

[1] [21] analyzes what Witty's design implies about its author.

[2] The main body of the Witty worm, including the initial pad required to cause the buffer overflow, fits in 675 bytes. However, the worm picks a larger packet-size, as shown in line 5 of Fig. 2, and pads the tail of the packet with whatever is on the stack, presumably to complicate the use of static filtering to block the contagion.

[3] Since $m = 2^{32}$, the modulo operation is implemented implicitly by the use of 32 bit registers and disregarding their overflow during arithmetic operations.

[4] The only related case of which we are aware was the Melissa email virus [3], where the author posted the virus to USENET as a means of initially spreading his malcode, and was traced via USENET headers.

[5] We ignore infectees that contributed < 20 packets.

[6] We can attribute the missing 5 Mbps to other, ever-present "background radiation" that is a constant feature at such telescopes [15].

[7] The distortion is not static but evolves with the spread of the worm. By tracking changes in the slope of the upper envelope, we can infer the value of the distortion against time throughout the period of activity of the worm.

[8] Recall that the effective bandwidth of most infectees is much lower than the access bandwidth, indicating heavy loss in their generated traffic.

Collaborating Against Common Enemies

Sachin Katti *MIT*   Balachander Krishnamurthy *AT&T Labs–Research*   Dina Katabi *MIT*

Abstract

This paper presents the first wide-scale study of correlated attacks, i.e., attacks mounted by the same source IP against different networks. Using a large dataset from 1700 intrusion detection systems (IDSs), we show that correlated attacks are prevalent in the current Internet; 20% of all offending sources mount correlated attacks and they account for more than 40% of all the IDS alerts in our logs. We also reveal important characteristics of these attacks. Correlated attacks appear at different networks within a few minutes of each other, indicating the difficulty of warding off these attacks by occasional offline exchange of lists of malicious IP addresses. Furthermore, correlated attacks are highly targeted. The 1700 IDSs can be divided into small groups with 4-6 members that do not change with time; IDSs in the same group experience a large number of correlated attacks, while IDSs in different groups see almost no correlated attacks. Our results have important implications on collaborative intrusion detection of common attackers. They show that collaborating IDSs need to exchange alert information in realtime. Further, exchanging alerts among the few fixed IDSs in the same correlation group achieves almost the same benefits as collaborating with all IDSs, while dramatically reducing the overhead.

1 INTRODUCTION

In this paper, we study *correlated attacks*, which we define as attacks mounted by the same source IP against different networks. Currently, about 30,000 new machines are compromised daily [25], and then used to launch attacks on other parts of the Internet. In many cases, the same machines are involved in multiple attacks against different networks [25]–i.e., correlated attacks. In addition to being an Internet phenomenon worthy of careful study, correlated attacks are important for collaborative intrusion detection. The intrusion detection system (IDS) at a network can exchange information about recent alerts and offending IPs with other IDSs. Future packets from suspicious source IPs can be flagged to be dropped or scrutinized. Such collaboration is most effective when it happens between networks experiencing correlated attacks.[1]

We present the first large scale empirical investigation of attack correlation in the Internet. We analyze logs from 1700 IDS/firewalls deployed in US and Europe. Our data is rich; in addition to sanitized logs from DSHIELD [2] and multiple universities, it contains *detailed* attack logs from 40 IDSs maintained by a Tier-1 provider to protect its customer networks. The logs cover 1-3 months, and a big chunk of the IP address space. In contrast to prior work, which has focused on the design of collaborative intrusion detection systems [28, 29, 21, 9, 26, 22], we address the following two questions:

- *How prevalent is attack correlation in the current Internet?* Although collaboration to detect common attackers seems plausible, there is no quantification of the potential benefits. Measurements of the frequency with which different networks become victims of a common attacker, the types of shared attacks, and the quantity of the resulting IDS alerts are important to gauge whether collaboration is worth the effort.
- *How can an IDS pick trusted and effective collaborators?* Allowing IDSs to exchange alerts to collaborate against common attackers requires addressing two issues: overhead and trust. Exchanging alert data with thousands of IDSs in realtime is a resource intensive task. Hence, an IDS needs to pick its collaborators intelligently to minimize the overhead and maximize the utility of the collaboration. Furthermore, two networks need to establish trust before they can exchange IDS data. Otherwise, a network cannot ensure the information it receives is not maliciously manipulated to make certain IP addresses look as attackers. Also, it cannot ensure that the information it provides will not leak internal vulnerabilities to malicious entities.

Our study results in 4 major findings.

(a) The extent of attack correlation: Correlated attacks are prevalent in the Internet; 20% of the offending IP sources attack multiple networks, and these common attackers are responsible for 40% of the total alerts in our dataset. Further, shared attackers attack different networks within a few minutes of each other, emphasizing the advantage of realtime IDS collaboration, as opposed to sharing attack logs offline.

(b) Reducing collaboration overhead by exploiting correlation structure: We analyze the spatial structure of attack correlation. We discover that the 1700 IDSs in our dataset can be divided into small groups of 4-6 members (about 0.4% of the IDSs in our set); IDSs in the same group experience highly correlated attacks, whereas IDSs in different groups see uncorrelated attacks. Collaborating with only IDSs in the same correlation group achieves the same utility obtained from collaborating with all IDSs, while dramatically reducing the collaboration overhead.

The small correlation groups seem to arise from recent attack trends. In particular, victim sites in the same group may be on a single hit list, or might be natural targets of a particular exploit like the Santy worm which attacked popular phpBB discussion forums scoured from search engines. We examined the correlated attacks in each group for cases where full attack details are available. Indeed, each group seems to be characterized by a specific attack type, e.g., there are SMTP groups, NT groups, IIS groups. This indicates that targeted attacks create small correlation groups of sites that run particular software/services.

(c) Scalable Trust Establishment: Our measurements reveal that correlation groups are fairly stable and their membership persists for the duration of the dataset (1-3 months). Thus, each network needs to collaborate with only 4-6 *fixed* networks in its group. The small number of IDSs in a group and their persistent membership allows a network to check their credibility offline and establish trust using an out-of-band mechanism such as legal contracts or reputation.

A network still needs to learn who is in its correlation group. This service can be provided by a few trusted nonprofit organizations, like CERT [1] and DSHIELD [2], or commercial entities. They receive sanitized alert data, (containing only time and offending source IP), from participating networks, analyze it for attack correlation, and inform the participating networks about others in their correlation group. The process is scalable because correlation groups are persistent for long intervals (months) and do not need frequent updates. Indeed, DSHIELD already has the means to provide this service to its participant networks.

(d) The importance of picking the right collaborators: We provide rough estimates of the overhead and detection capability obtained via different choices of collaborating IDSs. We focus on collaboration to quickly blacklist malicious IP sources. Using a trace driven simulation, we compare the following schemes: (1) correlation-based collaboration (CBC), where each IDS collaborates with only IDSs in its correlation group; (2) random collaborators, where an IDS collaborates with the same number of IDSs in its correlation group but picks the identity of its collaborators randomly. (3) local detection with no collaboration; (4) collaboration with all IDSs in the dataset;

The results of our evaluation emphasize the importance of picking the right collaborators. Mainly:

- CBC has almost as good detection capability as collaborating with all IDSs, but generates less than 0.3% of the traffic overhead. It detects 95% of the attackers detected by collaborating with all IDSs and reduces alert volumes by nearly the same amount.
- In comparison with local detection, CBC increases the number of detected common attackers at an IDS by 30% and speeds up blacklisting for about 75% of the common attackers. As a result of the blacklisting, correlation-based collaboration reduces the size of the log that the administrator has to examine by an additional 38%.
- Replacing the IDSs in the correlation group by random collaborators reduces the detection capabilities dramatically and does not add much beyond local detection.

Table 1 defines the terms used in this paper.

2 DATASET AND METHOD

2.1 Dataset

Our dataset is both large and rich. We use logs collected at 1700 different IDSs deployed in US and Europe. Our logs can be divided into 3 distinct sets based on their origin: (1) 40 IDSs on different networks in a Tier-1 ISP; (2) DSHIELD Logs; (3) University logs. The logs cover periods of 1-3 months. They span a relatively large fraction of IP address space. In addition to a /8 ISP space, the DSHIELD data contain logs from many /16 and /24 networks. This is the first studied dataset of its size that provides detailed alert information from deployed IDSs in the commercial Internet. Table 2 provides a summary description of the dataset. A detailed description is below.

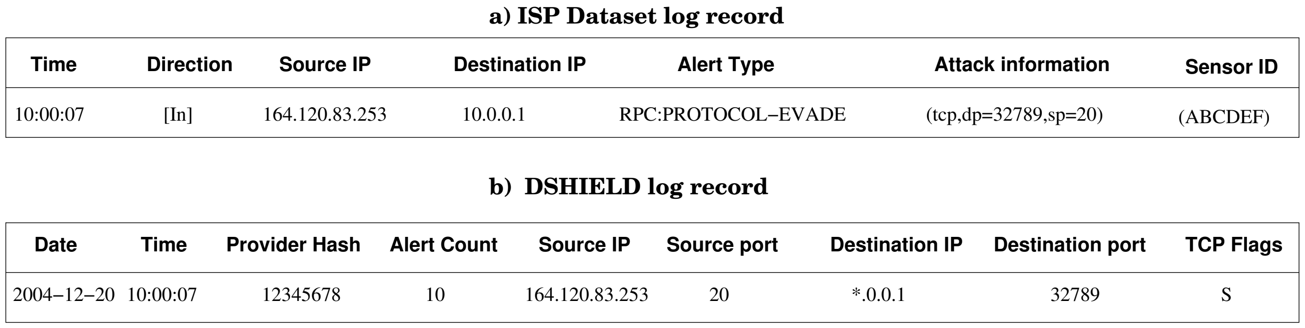
(a) ISP Logs: We have logs from 40 IDSs deployed in a large ISP with a /8 address space. The IDS boxes protect different customer networks and span a large geographic area, but they are all administered by the ISP and hence have identical characteristics and signature sets. The signature set is large and diverse con-

Term	Definition
Correlated Attacks	Two attacks are correlated if they are mounted by the same source IP.
Alert	An alarm raised by a sensor when it encounters a suspicious event, e.g. a packet or set of packets that contain a known exploit.
Correlated IDSs	Two IDSs are said to be correlated if more than 10% of their attacks are correlated.
Correlation Group of IDSs	A set of IDSs whose attacks are highly correlated.
Correlation Vector of IDS i	is $\vec{v}_i = (v_{i1}, ..., v_{ij}, ..)$, where $v_{ij} = 1$ if $j \in$ correlation group of i, and otherwise $v_{ij} = 0$.
Blacklist	A list of suspicious IP addresses whose packets are dropped or given unfavorable treatment.

Table 1: Definitions of terms used in the paper

	ISP dataset	DSHIELD	University datasets
# of IDSs	40	1657	3
Address space	Class A	5 Class B, 45 Class C and several smaller networks	2 Class B, 1 Class C
Period	July 1 - August 30, 2004 Dec. 15, 2004 - Jan. 15, 2005	Dec. 15, 2004 - Jan. 15, 2005	Dec. 15, 2004 - Jan. 15, 2005
Richness	Detailed alerts, unanonymized	Dest. IP addresses anonymized	Detailed alerts, unanonymized
Avg #alerts/day/IDS	40000	15000	30000

Table 2: Description of the 3 datasets



a) ISP Dataset log record

Time	Direction	Source IP	Destination IP	Alert Type	Attack information	Sensor ID
10:00:07	[In]	164.120.83.253	10.0.0.1	RPC:PROTOCOL–EVADE	(tcp,dp=32789,sp=20)	(ABCDEF)

b) DSHIELD log record

Date	Time	Provider Hash	Alert Count	Source IP	Source port	Destination IP	Destination port	TCP Flags
2004–12–20	10:00:07	12345678	10	164.120.83.253	20	*.0.0.1	32789	S

Figure 1: Log records for the ISP dataset and the DSHIELD dataset. The ISP dataset also has packet headers for each log record. The DSHIELD dataset has the destination IP anonymized.

sisting of over 500 different alerts. The logs contain full unanonymized packet headers for all suspicious packets, as shown in Figure 1a. Hence unlike the DSHIELD data described below, we have access to the offending packet as well as the nature of the offense. The logs cover two separate periods: one period from July 1 to August 30, 2004 and the other from December 15, 2004 to January 15, 2005. The data exhibits a large amount of variation in the kind of attacks seen (over 100 different attack types) as well as the distribution of attacking IP addresses (over 100000 unique source addresses) and 40000 alerts/day/IDS.

(b) DSHIELD Logs: DSHIELD is a global repository set up as a research initiative as part of the SANS institute. Participating organizations provide IDS/firewall logs, which DSHIELD uses for detection and analysis of new vulnerabilities, and blacklist generation. Since the IDS systems which participate in DSHIELD employ widely varying software, DSHIELD uses a minimal record format for its logs and scrubs the high order 8 bits of the destination IP address, as shown in Figure 1b. The entities participating in DSHIELD vary in size from several Class B networks to smaller Class C networks and are distributed throughout the globe [28, 2]. The logs are of substantial size with nearly 15000 alerts/day/IDS. We have collected DSHIELD logs from 1657 IDSs for the period from Dec. 15, 2004 to Jan. 15, 2005 corresponding to the ISP dataset.

(c) University Logs: Finally, we collect a set of logs from IDS/firewall systems deployed at 3 universities U1, U2 and U3. Of these we have access to raw data complete with packet headers and nature of offense detected in U1. The second university U2 provided us with logs from running the Bro IDS [19], but with protected addresses anonymized. The signature set deployed is different and the alerts consist mostly of scans of IP addresses as well as port-scans. The third university U3 provided us with firewall logs which consisted of blocked connection attempts. The University logs generate 30000 alerts/day/IDS on the average.

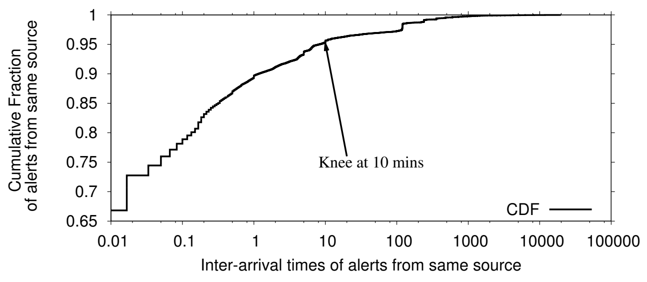


Figure 2: CDF of inter-arrival times of consecutive alerts from a source in minutes. The CDF is taken over the inter-arrival times. 95% of consecutive alerts from a source arrive within 10 minutes of each other, the rest are separated by several hours.

A few limitations are worth mentioning. Except for the ISP logs, the other IDSs in the logs are largely independent. We do not have access to their configurations, and hence we do not know the signature sets they employ, or even the platforms they use. This means that some of the attack correlation may be hidden because of differences between IDS signature sets. Second, we do not have information about the nature or the business of the protected networks, and thus cannot tell whether these issues play a role in attack correlation.

2.2 Method

Before studying attack correlation, we clean the data from obvious false positives, and analyze it to find a meaningful definition of the term "attack correlation".

2.2.1 *Filtering*

IDS logs are prone to flooding with alerts, many of which are innocuous alarms. For example, the ISP and University data sets contain innocuous alarms triggered by misconfigurations, P2P applications like eDonkey, malformed HTTP packets etc. Many of these were already flagged as false positives by the security administrators of the ISP. Since these are not actual attacks, they do not help in detecting attack correlation among different sites. Hence we filter out known false positives from the ISP and universities logs. We consider all the remaining alerts to be parts of valid attacks. Of course we cannot do this for the DSHIELD dataset, since we do not know the nature of the alert.

2.2.2 *Attack Durations*

To carry out this study, we need to extract attacks from IDS logs. We consider a stream of suspicious packets from the same source to an IDS with an inter-arrival smaller than 10 minutes as an attack. Below we explain why a separation window of 10 minutes is reasonable.

To find a meaningful separation window, we plot a CDF of inter-arrival times of consecutive alerts from the same source at an IDS in Figure 2. The CDF shows that 90% of the alerts from a source arrive within a minute of each other, these are likely to belong to the same attack event. The knee in the CDF happens at 10 minutes, inter-arrival times larger than 10 minutes are spread out to several hours. We pick 10 minutes as the window because about 95% of the alerts from the same source arrive separated by less than 10 minutes and the other 5% have widely-spread interarrivals.

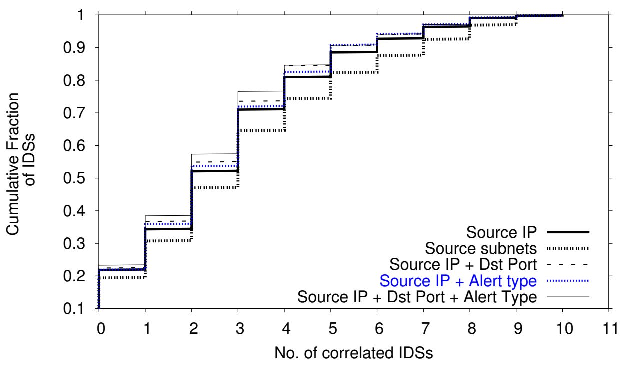


Figure 3: CDF of the size of the correlation groups for different definitions of attack correlation for the ISP and U1 datasets. The CDFs are taken over the IDSs. They show that the correlation is insensitive to the additional information obtained from the alert type and port, and can be discovered based solely on source IP.

2.2.3 *Defining Attack Correlation*

How should one define attack correlation? Should all fields in the alerts received at different IDSs be the same, or is it enough to consider one or two fields? Furthermore, how long can the interval between the two attacks at two different IDSs be for them to be still considered correlated?

Attack correlation can be parameterized by the set of correlated header fields and the time window used to compute the correlation. *We define two attacks to be correlated if they share the source IP address and start within 10 minutes of each other.* Both choices are based on detailed analysis of the data that showed almost no sensitivity to including additional fields in the correlation beyond the source IP and using time windows larger than 10 minutes. Below we describe this analysis in detail.

(a) Picking the correlation fields: Defining attack correlation based on the destination IP address is not useful since attacks seen by a particular IDS will have their destinations in the local network. Also the source port is likely to be picked randomly and is not useful for defining attack correlation.

We consider the following definitions of correlated attacks: 1) source based, 2) source and the destination port combined, 3) source and alert type combined, 4) source, alert type, and destination port combined, 5) and source subnet based. We conduct this analysis for the ISP dataset and the U1 datasets, for which we have access to all these fields.

Since our main interest is to find who is correlated with whom, we consider how different attack correlation definitions affect the size of the correlation group of a IDS (see Table 1). Correlated groups are explained further in §3, but for the purposes of this analysis they are simply the set of IDSs with which a particular IDS shares correlated attacks.

Figure 3 plots the cumulative distribution functions (CDFs) of the size of the correlation group of an IDS. Different CDFs correspond to different correlation fields. The figure shows that, except for the CDF for source subnets, all the other CDFs are very close together. Classification based on the attacking source subnet results in slightly higher correlation, but the difference is not substantial. Further, classifying based on source subnet carries the danger of blacklisting an entire subnet resulting in innocent sources being blocked. Since including extra fields in the definition of correlation in addition to the source IP has no significant impact on the correlation CDF, we define attack correlation based solely on the similarity of the offending source IP.

The above leads to an interesting result: performing attack correlation analysis requires minimal information, namely attack time and offending source IP.

(b) Picking the maximum time window between correlated attacks: Unless stated differently, a 10 minute window is used for determining correlated attacks at different IDSs. We tried different time windows in the $[5, 30]$ minutes range. Windows less than 10 minutes resulted in decreased attack correlation while there was not much difference for windows greater than 10. Hence we picked the minimum window possible i.e., 10 minutes. Thus, if two attacks at two IDSs start within 10 minutes of each other, then they are considered correlated.

(c) Correlation threshold: We say that two IDSs are correlated if more than 10% of their attacks are correlated. We justify the threshold below. We compute the CDF of correlation taken over all IDSs with non-empty groups (i.e., IDSs that are correlated with at least one other IDS). For 90% of the IDS, the correlation (percentage of correlated attacks w.r.t all attacks) was higher than 10% ranging upto 57%. For the remaining 10% of the IDS, the correlation was slightly higher than 0%. Such small values are due to a few attacks being shared and do not reflect any significant correlation between the two IDSs.

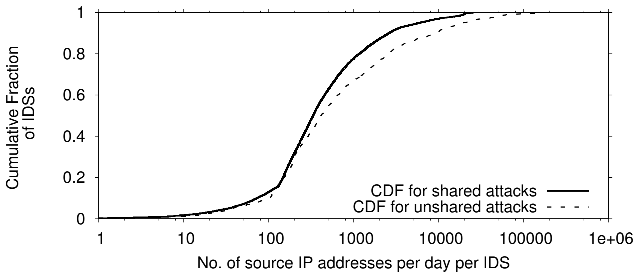


Figure 4: Prevalence of common attackers. Figure shows the CDFs of the average number of common attackers and local attackers per day per IDS. A common attacker is a source IP that is flagged as suspicious at two or more IDSs. 90% of the studied IDSs see more than 100 common attacking IPs per day. The average number of common attacking IPs at an IDS is about 1,500 while the maximum can be as large as 25,000.

3 EXTENT OF ATTACK CORRELATION

3.1 Do IDSs see common attackers?

A common attacker is an IP address that generates alerts at two or more IDSs. We compute the average number of common and uncommon attacking IP addresses for each IDS per day. Figure 4 compares the CDF of common attackers with the uncommon/local ones. The CDF is taken over all IDSs. The graphs show that on average an IDS sees 1500 shared offending IPs per day, and 6000 unshared offenders. Thus, about 20% of the suspicious source IP addresses observed at an IDS are also seen at some other IDS in the dataset. These common source IP addresses account for 40% of all alerts in the logs. *Thus, correlated attacks happen quite often and constitute a substantial fraction of all attacks.*

3.2 How many victims does a common attacker attack?

The previous section quantified how many source IP addresses at each IDS are common attackers, here we focus on the number of victims of a common attacker. Figure 5 plots the CDF of the number of IDSs targeted by a common attacker. The CDF is taken over all common attacker IPs. On the average, a common attacker appears at 10 IDSs, which is about 0.6% of all IDSs in the dataset. The high average of 10 victims seems to comply with recent trends in using botnets to mount multiple attacks against many target networks [25].

3.3 Time Between Correlated Attacks

How long does it take a common attacker before he attacks the next network? If this time is long then

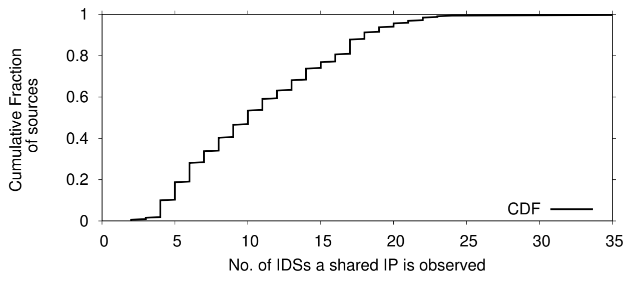


Figure 5: Figure shows the CDF of the number of different IDSs targeted by a common attacker. Common sources are detected at 10 different IDSs on the average, implying that such sources are employed to mount a large number of attacks at different victims.

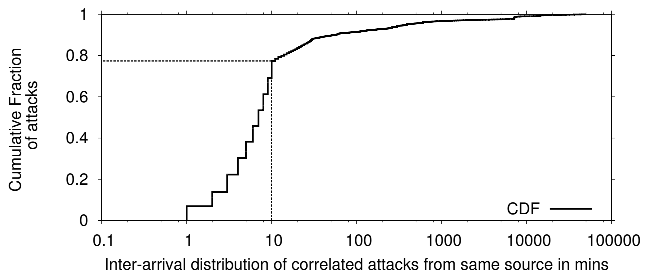


Figure 6: Figure shows the CDF of the interarrival times of correlated attacks at different IDSs. More than 75% of the correlated attacks arrive within 10 minutes of each other. This emphasizes the need for realtime exchange of attack data.

the exchange of alert data can be offline, but if it is short then effective collaboration against common attackers requires realtime exchange of information. We compute interarrival times of attacks from the same source at multiple IDSs, i.e., the difference between when the first time the attacker is observed at different IDSs. Figure 6 shows the CDF of these interarrival times. More than 75% of the time, a common attacker attacks the next IDS within 10 minutes from the previous IDS. Attackers therefore mount multiple attacks within a span of a few minutes, suggesting that collaborative detection of such attackers has to be in realtime.

4 ATTACK CORRELATION STRUCTURE

Why is the structure of attack correlation important? Since correlation is prevalent, it would be beneficial for IDSs to collaborate to speedup the detection of common attackers. However, in §3.3, we have shown that common attackers attack their victim networks within a few minutes of each other. Thus, to effectively collaborate against common attackers, the IDSs need to exchange information in realtime. An IDS in our dataset generates on average 1500 alerts/hour. Exchanging alerts in realtime with thousands of IDSs creates an unacceptable overhead. Thus, we are interested in finding how many collaborators each IDS needs to have in order to achieve the benefits of collaboration without incurring much overhead. To answer this question, we examine the spatial and temporal structures of attack correlation, i.e., how many IDSs are usually correlated with each other and how often does the set of IDSs a particular IDS is correlated with change over time?

4.1 Correlated IDSs

For the objective of detecting common attackers, an IDS benefits from exchanging alerts with only those IDSs whose attacks are correlated with its own. We call this set of IDSs its correlation group. If correlation groups are small, i.e., much smaller than all IDSs, then by focusing only on the IDSs in its correlation group, an IDS can achieve most of the benefits of the collaboration at little overhead.

We plot in Figure 7 the CDF of the number of IDSs with which an IDS is correlated (i.e., the size of its correlation group) for all 1700 IDSs in our dataset. We consider two cases: simultaneous correlation, in which two attacks are correlated if they share the same source IP and happen within 10 minutes of each other, and general correlation, in which two attacks are correlated if they share the source IP. The former helps detect distributed attacks, while the latter helps detect malicious sources which should be blacklisted. General correlation is by definition greater than simultaneous correlation. The figure shows that on average each IDS is correlated with $4-6$ other IDSs, i.e., less than 0.4% of the total number of IDSs. Further, 96% of the IDSs are correlated with less than 10 IDSs.

Note that the plots for simultaneous and general correlation are fairly similar. Though the average number of IDSs with which an IDS shares attacks increases to nearly 5, the CDF does not change much. Again, this shows that when correlated attacks happen at different locations in the Internet, most likely they happen with a short period.

4.2 Persistence of IDS Correlation

We would like to examine how often the correlation group of an IDS changes. If the membership of the correlation group of an IDS is stable then each network can spend the time to identify its correlation group offline. Once the correlation group is identified, the actual exchange of alerts is done in realtime. On the other hand, if the members of an IDS' correlation group keep changing over short intervals, collaboration will be hard as it requires re-examining attack correlation and deciding in realtime whether to collaborate.

We need to define a measure of how a group of IDSs is changing. We assign the IDSs consecutive IDs. For

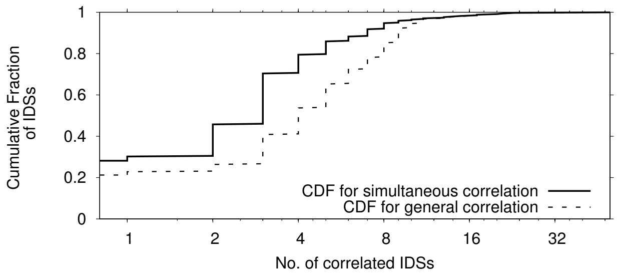


Figure 7: Cumulative Distribution of the number of IDSs with which any IDS exhibits correlation for all 3 datasets. Figure shows most IDSs are correlated with 4-6 (among 1700) IDSs with the average being slightly higher than 4.

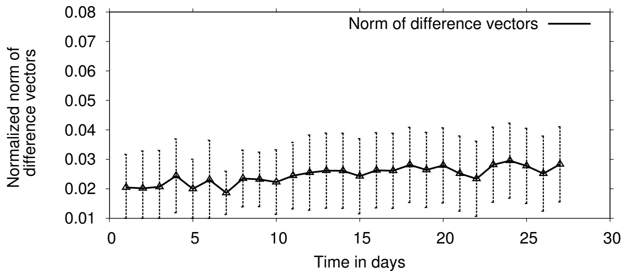


Figure 8: Figure shows that the group of IDSs which experiences attacks correlated with attacks at a particular IDS does not change for the duration of our 3 datasets (about a month). The y-axis is the normalized difference in the correlation vector defined in Equation 1.

each IDS i in our dataset, we create a correlation vector $\vec{v}_i(n)$ whose length is equal to the total number of IDSs in the dataset. We set $v_{ij}(n) = 1$ if IDS i is correlated with IDS j, and 0 otherwise based on the alerts they generate on day n. For example, $\vec{v}_i(16) = (0, 1, 1, 0, 1, 0, ..., 0)$ means that IDS i and IDSs 2,3, and 5 see correlated attacks on the 16th day in our dataset.

The difference vector for two days for a given IDS is the vector obtained by subtracting the corresponding correlation vectors for those days. For example, the difference $v_i(17) - v_i(0)$ indicates how the correlation group of IDS i changes over a period of 17 days, starting on day 0 in our logs.

We measure the persistence of attack correlation as a function of time using the following metric:

$$f_{m-n} = \frac{1}{N} \sum_i \frac{||\vec{v}_i(m) - \vec{v}_i(n)||}{||\vec{v}_i(n)||}, \qquad (1)$$

where $N = 1700$ is the number of IDSs; v_i is the correlation vector of IDS i; and $||\vec{v}||$ is the Euclidean norm of the vector. Thus, f_{m-n} is the average change in the norm of the correlation vector between day n and day m where $m > n$, normalized by the size of that vector.

Figure 8 plots our measure of the difference in attack correlation f_i as a function of time in days along with the standard deviation. It shows that, the correlation vector does not change significantly with time. In particular, on average the correlation vector changes by less than 0.025 of its original value over a period that spans a whole month. The insignificant change shows that *correlation happens consistently with the same group of IDSs and is persistent over time.*

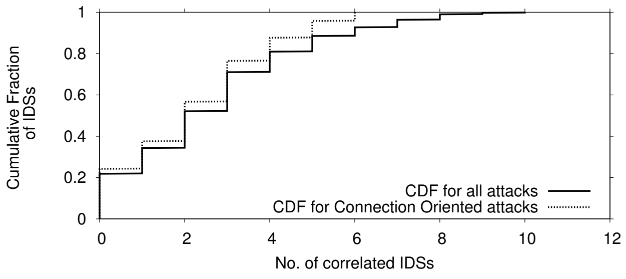


Figure 9: Comparison of attack correlation among connection-oriented attacks and all attacks for the ISP dataset. The figure plots the CDF of the number of IDSs that experience correlated attacks to a particular IDS. The two CDFs are very close indicating that our results are robust against source spoofing.

4.3 Robustness to Source Spoofing

The correlation shown above considers all attacks, including those which could be from spoofed source addresses. Intuitively, one would expect that source spoofing does not affect the correlation structure as it is usually done randomly, and thus unlikely to create a well-defined structure. In order to estimate the effect of spoofed sources on our results we divide the logged attacks into two classes:

- *Connection oriented attacks*: Attacks which require establishing a TCP connection. This includes most non-flooding attacks and application layer attacks (e.g SQL server, MS IIS server etc) and formed 68% of all attacks.
- *Connectionless attacks*: Attacks which get flagged due to incomplete TCP connection attempts or those which do not require a TCP connection. (e.g. SYN floods, UDP packet floods etc).

We can perform this classification only on the ISP data and one of the Univ logs (U1). The rest of the logs do not contain the necessary information. Connection-oriented attacks should not have spoofed IP addresses since they require the attacking machine to respond to the TCP ACKs sent by the victim.

Figure 9 compares the correlation exhibited by the connection oriented attacks to that exhibited by the combination of all attacks. The figure plots the CDF of

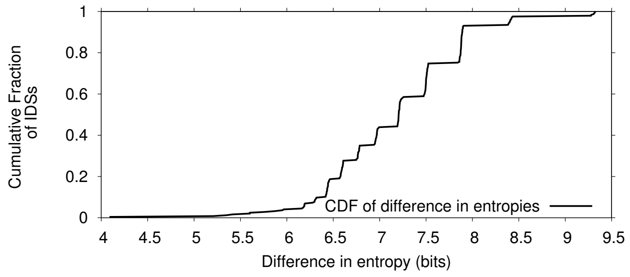


Figure 10: Figure shows that the set of IDSs with which an IDS is correlated is far from random. We compare the distribution of correlated IDSs in our dataset with that generated by having each common attacker target a small random set of IDSs. The difference in entropy between random targeting and the empirical data is plotted. The empirical distribution has, on average, 7.2 bits less entropy than the one generated by random targeting; correlated IDSs are therefore far from random.

the size of the correlation group for each IDS for each kind of attack. *The figure shows that the two CDFs are very close, indicating that the correlation structure is highly robust to source spoofing.* Similarly, we have performed the correlation persistence test in §4.2 on connection oriented attacks and found the results to be compatible with those in §4.2.

4.4 Is the structure due to random scans?

The fact that each IDS in our dataset shares attacks with only a small and persistent set of IDSs is intriguing. Why do certain IDSs share attacks? Before answering this question, we would like to do an additional test to ensure that the spatial structure of attack correlation is not random. Suppose each worm or attacker picks for victims a random subset of all destinations, could this be responsible for generating the attack correlation structure we see in the data? The test described below shows that the answer to the this question is "no". The correlated attacks we see are likely targeted attacks, i.e., the victims are not randomly chosen; the same group of correlated victim networks are chosen repeatedly, probably because they are on one hit list circulating among the attackers, or because they run the same software (as in the case of the Santy worm [6]).

We consider the distribution of IDSs with which a particular IDS is correlated. We compare this distribution in our data with the corresponding distribution generated by random targeting. We simulate random targeting as follows. We pick an IDS, i, and look at all of its correlated attacks. For each correlated attack, we replace the set of IDSs with which IDS i shares this attack with a random set of IDSs of the same size. We repeat this process for each attack at IDS i. For each IDS j, where $j \neq i$, the number of correlated attacks with i, after proper normalization, represents the probability that IDS j is correlated with IDS i. We compare this probability distribution in our data with the one generated by random attack targeting. In our data, this distribution is highly biased, i.e., an IDS i is correlated with a few other IDSs and uncorrelated with the rest of IDSs. Since we are interested in measuring how far our data is from random targeting, we compare the entropy of the two distributions. The entropy of the distribution of a random variable X is:

$$H(X) = -\sum_{x_i} P(x_i) log(P(x_i)). \qquad (2)$$

This analysis is repeated for each IDS and the difference in entropies are computed for each IDS. Figure 10 shows the CDF of these entropy differences. The figure shows that the set of IDSs with which an IDS is correlated is far from random. It shows that the empirical distribution has, on average, 7.2 bits less entropy than the one generated by random targeting. Note that number of IDSs in our system is 1700, hence the maximum entropy is 10.73 bits. The difference in entropy is also bounded by the same value. Thus, an entropy difference of 7.2 bits is very high, which shows that the set of correlated IDSs in our data is far from random.

4.5 Origin of IDS Attack Correlation

So why two IDSs share correlated attacks? We investigate two possible reasons: 1) closeness in the protected IP space, 2) similarity in the software and services run on the two sites. Our results show that the latter is the likely reason of attack correlation between two IDSs.

(a) Closeness in IP space: Some attackers employ scanning techniques to discover vulnerabilities. They start from a randomly selected IP and then scan sequentially. If the scanned address spaces belong to different sites, the IDS at the respective sites are likely to show attack correlation. Thus, closeness in the IP space could be a reason for attack correlation.

We compute the distance between two prefixes P_1 and P_2 of equal length as the decimal value of the bitstring produced by taking XOR of P_1 and P_2. If the prefixes are of unequal length, the shorter prefix is bitshifted to the left to equalize the lengths. The distance in IP space between two IDSs i and j, D_{ij}, is defined as the IP distance between their protected address prefixes. Also for each IDS pair we generate the vector of correlation $\vec{C}_{ij}$, where c_{ij} is the percentage of attack at i which are correlated with some attacks at j. If proximity in the IP space is a reason for attack correlation, then the more the distance between IDSs i and j is, the less likely they share correlated attacks–i.e.,

$\vec{D}_{ij}$ and $\vec{C}_{ij}$ should be inversely correlated. Thus, we compute the cross correlation between these two vectors.[2] Note that a cross correlation around zero means independence. Figure 11 plots the cross correlation between attack correlation and distance in IP space. The x-axis is the IDS id. Note that *the correlation with IP space hovers around zero, indicating that attack correlation is independent from the distance in IP space.* Thus, having nearby IP prefixes does not have a visible impact on sharing correlated attacks.

(b) Similarity in Software and Services: Small correlation groups may be due to recent attack trends. In particular, two IDSs may share correlated attacks because they are on a single hit list, or they run software or a service that is targeted by the common attacker. For example, the Santy worm uses a vulnerability in popular phpBB discussion forum software to spread and uses a search engine to find vulnerable servers [6].

Unfortunately, except for the university logs (U1), we do not know the identity of the protected networks, the type of software they run, or the services they provide, and hence cannot check attack correlation against that information. Instead we perform two indirect tests.

First, we have examined the correlated attacks in each group for the case of the ISP data where full attack details are available. Indeed, except for one correlation group, each group seems to focus on a specific shared attack, i.e., more than 60% of the correlated alerts in that group are of a particular type. There are SMTP groups, NT groups, IIS groups, etc. This should not be surprising as recent attacks obtain a list of networks that run a software with the targeted vulnerability via a search engine or other ways and send only to those sites [6].

Second, we try to indirectly infer the software and services run on the correlated networks by comparing the type of alerts they generate. We compute the distribution of alert types generated by each network and compare them against each other. We divide alerts into 13 broad categories: alerts due to attacks on DNS servers, web servers, ftp, RPC services, Windows Server 2003, servers running RPC, mail servers, servers using SQL (both MS and MySQL), telnet and ssh servers, attacks on routers, IRC servers, CIFS (SMB) servers and miscellaneous. We compute the fraction of alerts of each type in the IDS log. We consider this distribution to be characteristic of the network itself, and check whether attack correlation is correlated with correlation in this distribution.

We express the alert distribution in a vector $\vec{V_i}$ with 13 elements. For example, $\vec{V_i} = (0.03, 0.2, ...)$ means that 0.03% of the alerts generated by IDS i are of category 1, etc. We measure the distance between the alert distributions at IDS i and j by the difference $\vec{D}_{ij} = ||\vec{V_i} - \vec{V_j}||$, where $||.||$ is the Euclidean norm. Similarly to the analysis in §4.5(a), we compare $\vec{D}_{ij}$ with $\vec{C}_{ij}$, where c_{ij} is the percentage of attack at i which are correlated with some attacks at j. If similar software and services are reasons for attack correlation, then $\vec{D}_{ij}$ and $\vec{C}_{ij}$ should be inversely correlated. We compute the cross correlation between these two vectors. Note that a cross correlation around zero means independence. whereas a negative cross correlation means that an increase in the distance, $\vec{D}_{ij}$, is correlated with a decrease in attack correlation $\vec{C}_{ij}$. Figure 11 plots the cross correlation between attack correlation and our indirect measurement of the similarity of the software and services on the protected networks. Note that attack correlation is negatively correlated with our measure of the distance between the software and services on the protected networks–i.e., an increase in this distance results in a decrease in correlation. Thus, it seems that one origin of attack correlation across different networks is the similarity in the software and services they run.

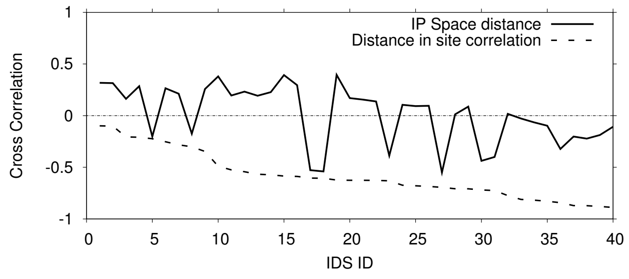


Figure 11: Cross correlation between attack correlation and: 1) distance in IP space, 2) an indirect measure of site's software and services. Figure shows that attack correlation is independent of closeness in IP space. In contrast, attack correlation seems to decrease with decreasing similarity between the software run on the protected networks. The figure is for the ISP and U1 datasets for which we have detailed alert logs.

5 SUMMARY OF EMPIRICAL RESULTS

The results of our study of attack correlation can be summarized as follows:

- Correlated attacks mounted by common attackers against multiple networks happen quite often. 20% of the unique sources in our dataset generate attacks at multiple IDSs, and common/correlated attacks account for an average of 40% of all attacks observed at an IDS.
- A network experiences attacks correlated with only a few other networks. On average an IDS shares

attacks with 4-6 other IDSs which is just 0.4% of the total number of IDSs, and 96% of the IDSs share attacks with less than 10 other IDSs.

- Attack correlation persists over time–i.e., the sets of IDSs that experience correlated attacks did not change for the duration of our study (1-3 months).
- Though we do not know all origins of attack correlation, our data shows that similarity in the software and services run on the protected networks plays an important role in making them endure correlated attacks.
- Common attackers tend to attack different networks within a few minutes of each other. Thus, there are considerable advantages for realtime sharing of alerts.
- Discovering the correlation group of an IDS (i.e., who shares with whom) requires minimal IDS-related information, namely attack time and offending source IPs.
- Our study of the correlation for connection-oriented attacks shows that the correlation groups and their member IDSs are robust to IP spoofing.

6 EFFICIENT COLLABORATION WITH TRUSTED PARTNERS

The major impediments to having independently administrated IDSs collaborate on detecting common attackers are: overhead and trust. Since common attackers attack different networks within a few minutes from each other, the IDSs need to exchange their alerts in realtime. But exchanging alerts with thousands of IDSs in realtime is impractical because of the resulting overhead, and the potential of having malicious IDSs incriminating innocent hosts or using the alert data to discover the vulnerabilities of other networks.

We exploit the structure of attack correlation to solve the above two problems.[3] We propose a correlation-based method for picking collaborators. By exchanging alert data with only those IDSs in its correlation group, an IDS minimizes the overhead of the collaboration while maximizing its chances of detecting common attackers. Furthermore, since the size of a correlation group is small and its membership is stable, an IDS can check using out-of-band mechanisms the reputability of each of the IDSs in its correlation group. It can use this information to decide whether to collaborate. If needed, the IDS can use legal contracts to enforce trust and privacy. If the IDSs choose to collaborate, they use a secure channel to exchange information so that eavesdroppers cannot snoop.

IDSs need to know which other IDSs are in their correlation group. We envision a number of non-profit organizations (like CERT and DSHIELD) and commercial entities that discover attack correlation across IDSs and report to each network the identity of the other networks in its correlation group. We call these entities Attack Correlation Detectors (ACD). A network may participate in one or more ACDs. The choice of ACD may depend on the number and types of networks participating in the ACD, its reputation, etc. The ACD occasionally collects logs from participant IDSs. The logs cover a particular period that can be as small as a single day randomly chosen by the ACD. The logs have minimal sensitive information. Each record in the log provides the following fields: (Time, Source IP, Packet Count). Our analysis in §*2.2.3* shows that these fields are enough for detecting attack correlation. The ACD performs the correlation analysis and informs each network of its correlation group, expressed as a list of the following records: (correlated IDSs, level of correlation). The correlation analysis is not intensive, the average time taken to analyze a days worth of logs is just 4 hours on an Intel Itanium 1.5 GHz SMP machine with 2 GB of memory. Further since IDS correlation is persistent over atleast a month, the analysis is repeated only after such long periods of time. Once organizations know their correlation group, they can independently decide with whom to collaborate, basing their decisions on the level of correlation and the identity of the peer network.

Integrating new IDSs and updating participant IDSs about changes in their correlation can be performed incrementally. A new IDS provides logs from the same collection point so that its correlation group can be found. Updates are incremental, since IDSs need to be informed only if their correlation group changes. Due to the persistence of membership in these correlation groups (a month or more), the update process can be performed in a lazy fashion with the cost amortized over long periods of time.

It should be noted that acting as an ACD is relatively simple. Indeed, DSHIELD already has the means to provide this service to its participant networks.

6.1 Discussion

(a) Scalability: Correlation-based collaboration ensures scalability by the small size of the groups and the persistence of correlation among IDSs across long timescales. In particular, over 96% of the IDSs in our dataset are correlated with less than 10 other IDSs. The overhead of setting up peering and exchanging information is therefore relatively small. Additionally, the persistence of correlation over months ensures the scalability of ACDs. The ACDs analyze correlation at these timescales, amortizing the cost of the analysis.

(b) Privacy: Recall that for discovering its correlation group an IDS provides the ACD with logs of attacking IP addresses, alert time, and packet count. Thus, none of the sensitive alert fields such as the attack type, the destination, and the destination port, are needed. Also the data is revealed only to the ACD and does not get published. On the other hand, privacy of the data exchanged with one's collaborators is provided largely because IDSs have the ability to independently decide which IDSs to collaborate with, and what to reveal. Further, the persistence of correlation allows the collaborators to use legal contracts to protect their data, if necessary.

(c) Protecting against spreading lies: An IDS that lies about its attackers to the ACD does not harm the system. Such lies are unlikely to be correlated with any of the attacks seen at other IDSs, even if they do, each IDS checks independently the credential of each of its collaborators before sharing any alert data with them. Lying to one's collaborators is unlikely as their reputations are carefully checked and information exchange is protected by legal contracts.

7 PICKING THE RIGHT COLLABORATORS

We present a rough evaluation of the overhead and the enhancement in detection capability obtained via various choices of collaborating IDSs for detecting correlated attacks. We compare the following 4 schemes for picking collaborators:

- **Collaborate With ALL IDS:** An IDS collaborates with all other IDSs in our dataset.
- **Correlation-Based Collaboration (CBC):** Each IDS collaborates with only those IDSs in its correlation group.
- **Random Collaboration:** An IDS picks a random set of IDSs to collaborate with. To ensure the comparison with CBC is fair, each IDS collaborates with as many IDSs as there are in its correlation group.
- **Local Detection:** in this scheme, detection is based on local alerts with no collaboration with other IDSs.

7.1 Blacklisting Malicious Sources

In order to compare the above schemes, we need to specify a protocol for exchanging alerts and processing the acquired information. We use the simple approach described below. This approach is not necessarily optimal, but it suffices to evaluate the relative benefits of the different methods of picking collaborators.

The IDSs collaborate to detect low rate attackers and speed up the detection of moderate rate attackers. Each IDS maintains a `Blacklisting Threshold` and a `Querying Threshold`. A source IP address is blacklisted when the number of suspicious packets from it crosses a `Blacklisting Threshold`. An IDS queries its collaborators when the number of malicious packets from a source IP address crosses the `Querying Thresholc` If the aggregate rate of the offending source at all collaborators exceeds the `Blacklisting Threshold` the source is blacklisted. Once a source is blacklisted it is set apart for further investigation and an alarm is triggered to all collaborators.

The time taken to blacklist a source depends on two factors; the rate at which the source is attacking as well as the chosen `Blacklisting Threshold`. In picking a particular threshold, there is an inherent tradeoff between false positive ratio and false negative ratio. A low `Blacklisting Threshold` will result in a high false positive ratio while a high threshold will miss many moderate rate attacks resulting in a high false negative ratio. The right value for the `Blacklisting Threshold` is site specific and should be picked to optimize the false negative and false positive ratios.

We use the ISP and U1 datasets to find a good value for the thresholds because these logs contain enough information to distinguish many cases of false positives. We set the `Blacklisting Threshold` to 1000 malicious packets/day because in our dataset, this rate results in a false positive ratio less than 1%. We set the `Querying Threshold` to 50 malicious packets/day. The `Querying Threshold` has to be substantially lower than the `Blacklisting Threshold`, but there is nothing special about the value of 50 packets/day. In reality, these thresholds will vary depending on the local sites configuration as well as the nature of the alert itself. The above thresholds seem reasonable for those IDSs in our dataset for which we have detailed attack information.

To simulate the attacks, we replay the traces in our datasets. We divide one month worth of traces into two equal parts, corresponding to 15 days each. The correlation groups are generated from one set (the training set), while the various schemes for picking collaborators are tested on the other set (the test set).

7.2 Detection Speedup

Figure 12 plots the time it takes to blacklist a source in each of the four approaches: CBC, Local Detection, Random Collaboration and Collaboration with All IDSs. The time to blacklist a source is defined as the time difference between the instant the source is blacklisted by some IDS and the instant the source was first detected by any of the collaborators. The plots are only for sources detected at more than 1 IDS, because localized sources always require the same time to detect under all four schemes. The malicious sources on the x axis are sorted according to their detection time by Local Detection. Note that for this figure, we set

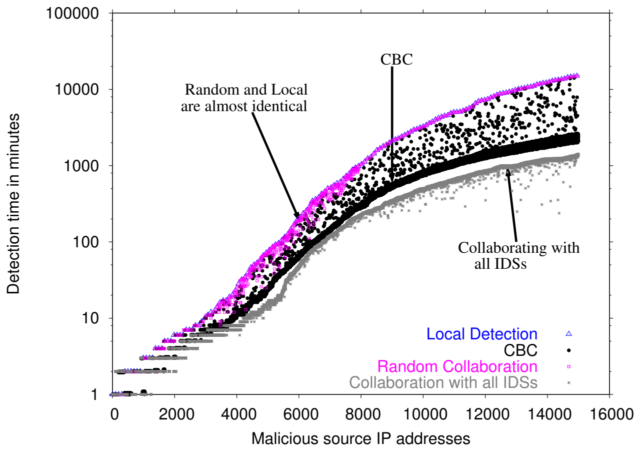


Figure 12: Comparison of time taken in minutes for blacklisting a shared malicious source for CBC, Local Detection, Random Collaboration and Collaboration with All IDSs. Short duration attacks (under 5 minutes) which number around 25% do not show significant difference, local detection works nearly as well. CBC performs nearly as well as collaboration with all IDSs in detecting longer duration, slower attacks. Random collaboration offers no benefit except for a few sources.

`Blacklisting Threshold` to a total of 1000 packets, rather than 1000 packet/day, so that each approach will eventually detect the malicious source.

The figure shows that, for fast sources which can be detected locally in 5 minutes or less, there is no significant difference among the four schemes. These form nearly 25% of all classified common attackers. The curves diverge for slower sources which take longer to blacklist locally. Random collaboration offers no benefit, i.e., the time taken to blacklist is the same as Local Detection except for a few sources. In contrast, CBC speeds up detection for about 75% of the studied sources, and performs nearly as well as collaborating with all IDSs. There are a small number around 5% of slower sources which take longer to detect in CBC because of them being correlated across IDSs which do not belong to each other's correlation group.

7.3 Overhead

In comparison with Local Detection, the speedup in detecting malicious sources is obtained at the cost of communication among the collaborators. The average query rates in CBC and Random Collaboration are fairly close. They both have an average of about 1.3 query/minute/IDS, with a standard deviation of 2.9. In comparison, collaborating with all IDSs has a very high overhead; the average query overhead is 454.9 query/minute/IDS, which is 2 orders of magnitude higher than CBC.

	CBC	Local Detection	Random Collaboration	All IDSs
Alert Reduction	73.44%	35.48%	37.77%	80.56%
Sources missed	5.02%	38.65%	36.69%	0%

Table 3: Comparison between 4 schemes for picking collaborators in terms of alert volume reduction and the number of malicious sources missed.

7.4 Effectiveness

Faster detection of malicious sources also results in significant reduction in alert volume. Table 3 lists the average reduction in alert volume from blacklisting under CBC, Random Collaboration, Collaborating with all IDSs and Local Detection. On average, CBC results in 73.44% reduction in alert volume (i.e., the size of the log that admin should examine). The value is close to the one obtained by collaborating with all IDSs, which is 80.56%. Local Detection, on the other hand, performs significantly worse; it reduces the alert volume only by 35.48%. There is no discernible difference between Local Detection and Random Collaboration, the reduction in alert volume is only marginally better at 37.77%. The above numbers are for all attacks, correlated and uncorrelated. Thus by being able to quickly detect correlated attacks, CBC reduces alert volume by a further 38% over Local Detection.

Table 3 also lists the fraction of correlated malicious sources missed by CBC, Local Detection, and Random Collaboration in comparison to Collaborating with all IDSs. A malicious source is missed if the scheme is unable to blacklist it due to incomplete information, though it is blacklisted all IDSs collaborate. CBC misses only 5.02% of the malicious sources, while Local Detection misses 38.65% of them. Random collaboration scheme is almost similar at 36.69%. These values depend on the thresholds used, but they demonstrate the order of magnitude improvement obtained in CBC and the insignificant difference between CBC and collaborating with all IDSs. In summary, CBC improves significantly over Local Detection. It increases the number of detected sources by 33%, and reduces the volume of alerts by an extra 38% beyond Local Detection. It performs almost as well as the collaborating with all IDSs. In contrast, a random choice of collaborators is as bad as not collaborating.

8 RELATED WORK

Several proposals exist for building collaborative and distributed intrusion detection systems [21, 9, 10, 26, 20, 23, 22, 8, 28], but none of them has studied attack correlation. Our work extends many of these proposals with a mechanism for picking collaborators, and maximizes the benefit of collaboration while limiting its overhead.

Early distributed intrusion detection systems collect audit data from distributed component systems but analyze them in a central place (e.g., DIDS [21], ISM [9], NADIR [10], NSTAT [26] and ASAX [8]). Recent systems have paid more attention to scalability (e.g., EMERALD [20], GrIDS [23], AAFID [22], and CSM [27]). We discuss a few of them below.

The Collaborative Intrusion Detection System [14] involves dynamic groups of nodes that rapidly change and exchange information. The set of nodes exchanging information is not constant and is changed continuously to cover all nodes in the system which limits its scalability. COSSACK [11], another collaborative response framework, is concerned more with alarm propagation than detection itself. DOMINO [28] relies on a hierarchy of nodes with different levels of trust and aims to exchange blacklist information. The nodes are placed such that IDSs protecting networks with close destination address spaces are close together.

The Distributed Intrusion Detection System (DIDS) [21] addresses system attacks across a network. Attacks such as doorknob, chaining, and loopback could be detected when data from hosts within a given network was combined under centralized control. Clever attackers could still subvert DIDS by reducing the volume of attacks for a given network.

EMERALD addresses intrusions within large separately administered networks [20]. EMERALD includes features for handling different levels of trust between the domains from the standpoint of a centralized system: individual monitors are deployed in a distributed fashion, but still contribute to a high-level event-analysis system. EMERALD appears to scale well to large domains. The Hummer project [15] focuses on the relationships between different IDSs (e.g., peer, friend, manager/subordinate relationships) and policy issues (e.g., access control, cooperation policies).

Finally, there has been work on specification and event abstraction to allow multiple IDS boxes to share attack information and collaborate on detection and protection [5, 24, 7].

Attack Measurements & Analysis: A lot of work has been done in characterizing attack characteristics. Yegneswaran et al. [28, 18] study the global characteristics of intrusions as well as Internet background radiation. Network telescopes are used to study DoS activity in [17]. Placement of blackholes in a distributed Internet setting for global threat detection is addressed in [3].

Analysis of Intrusion Alerts: GrIDS [23] collects traffic and connections data. It analyzes TCP/IP network activities using activities graphs and reports anomalies when activity exceeds an user specified threshold. Methods of discovering intent by correlating alerts from different IDSs are presented in [12]. Algorithms for sharing of alerts [13] in a privacy-preserving manner could be a future avenue of research. Alert correlation to reduce the number of alerts to be manually examined is discussed in [4]. Alerts are inserted into a relational database to be aggregated and the summarized alert is presented to the operator. These are orthogonal to our work and can be easily integrated.

9 CONCLUDING REMARKS

We have presented the first wide-scale study of attack correlation in the Internet, i.e., attacks that share the source IP but occur at different networks. Our dataset, constituting of alert logs collected at 1700 IDSs, show that correlated attacks are fairly prevalent in today's Internet; 20% of all the attacking sources are shared attackers, and they are responsible for 40% of all alerts in our logs. Shared attackers attack different networks within a few minutes of each other, emphasizing the advantage of realtime collaboration between victim networks as opposed to sharing attack information offline.

Our results also show that the 1700 IDSs can be grouped into small correlation groups of 4-6 IDSs; two IDSs in the same correlation group share highly correlated attacks, whereas IDSs in different correlation groups see almost no correlated attacks. Furthermore, the correlation groups are stable and their membership persists for months. Though not conclusive, our analysis indicates that similarity in the software and services running on the protected networks causes their IDSs to show attack correlation.

Our empirical results have important implications for collaborative intrusion detection of common attackers. They show that it is quite important that each network/IDS picks the right collaborators. Exchanging alerts with thousands of IDSs in realtime is impractical because of the resulting overhead and the lack of trust between these networks. Using a trace-driven simulation, we show that picking at random a smaller and fixed set of collaborators has almost no benefits beyond local detection. In contrast, collaborating with the 4-6 IDSs in one's correlation group has almost the same utility as collaborating with all 1700 IDSs in our dataset with 350 times less overhead.

Finally, we note that our results reflect the state of the Internet at the end of 2004 and the beginning of 2005. It is hard to predict the extent of attack correlation in the future and the continuous existence of correlation groups. Future research should investigate these characteristics and track their evolution.

10 ACKNOWLEDGMENTS

We would like to thank John Hardenbergh, Ben Leong, Harsha Madhyastha, Vyas Sekar and the anonymous referees for their comments; the Internet Storm Center for providing us the DSHIELD data; Ed Amoroso, Martin Arlitt, Tim Battles, Glenn Fowler, Patrick Haffner, Adam Hammer, Christopher Morrow, Manuel Ortiz, Dan Sheleheda, and Vinod Yegneswaran for their help with our project. Also, Katti and Katabi acknowledge the support of the National Science Foundation under NSF Career Award CNS-0448287. The opinions and findings in this paper are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Computer Emergency Readiness Team. http://www.us-cert.gov/.

[2] Distributed Intrusion Detection System. http://www.dshield.org/.

[3] E. Cooke, M. Bailey, D. Watson, F. Jahanian, and D. McPherson. Towards understanding distributed blackhole placement. In *The 2nd Workshop on Rapid Malcode (WORM) Fairfax, Virginia, October 29, 2004.*

[4] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *2002 IEEE Symposium on Security and Privacy.*

[5] D. Curry and H. Debar. Intrusion detection message exchange format: Extensible markup language (xml) document type definition, 2001.

[6] F-SECURE. F-secure virus descriptions : Santy. http://www.f-secure.com/v-descs/santy_a.shtml/.

[7] B. Feinstein, G. Matthews, and J. White. The intrusion detection exchange protocol (idxp), 2003.

[8] N. Habra, B. L. Charlier, A. Mounji, and I. Mathieu. ASAX : Software architecture and rule- based language for universal audit trail analysis. In *ESORICS*, 1992.

[9] L. T. Heberlein, B. Mukherjee, and K. N. Levitt. Internet security monitor: An intrusion detection system for large-scale networks. In *Proceedings of the 15th National Computer Security Conference*, 1992.

[10] J. Hochberg, K. Jackson, C. Stallings, J. McClary, and J. DuBois, D.and Ford. NADIR: An automated system for detecting network intrusions and misuse. In *Proceedings of Computers and Security 12(1993)3*, 1993.

[11] A. Hussain, J. Heidemann, and C. Papadopoulos. COSSACK: Coordinated Suppression of Simultaneous Attacks. In *DISCEX*, 2003.

[12] C. Krugel, T. Toth, and C. Kerer. Decentralized Event Correlation for Intrusion Detection. In *4th International Conference on Information Security and Cryptology 2001.*

[13] P. Lincoln, P. Porras, and V. Shmatikov. Privacy-Preserving Sharing and Correlation of Security Alerts. In *Usenix Security 2004, San Diego, CA.*

[14] M. Locasto and et al. Collaborative Distributive Intrusion Detection. In *CU Tech Report CUCS-012-04*, 2004.

[15] J. McConnell, D. Frincke, D. Tobin, J. Marconi, and D. Polla. A framework for cooperative intrusion detection. In *NISSC*, pages 361–373, 1998.

[16] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *INFOCOM*, 2003.

[17] D. Moore, G. M. Voelker, and S. Savage. Inferring internet Denial-of-Service activity. In *USENIX Security 2001.*

[18] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the IMC 2004.*

[19] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[20] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997.

[21] A. Snapp and et. al. Distributed intrusion detection system - motivation, architecture, and an early prototype. In *Proceedings of the 14th NCSC*, 1991.

[22] E. Spafford and Z. D. Intrusion detection using autonomous agents. In *Computer Networks, Volume 34*, 2000.

[23] S. Staniford-Chen and et. al. GrIDS – A graph-based intrusion detection system for large networks. In *19th National Information Systems Security Conference*, 1996.

[24] B. Staniford-Chen S.; Tung and D. Schnackenberg. The Common Intrusion Detection Framework (CIDF). In *Information Survivability Workshop, Orlando FL*, 1998.

[25] The Honeynet Project. Know your Enemy: Tracking Botnets. http://www.honeynet.org/papers/bots/.

[26] G. Vigna, S. Eckmann, and R. Kemmerer. The stat tool suite. In *In Proceedings of DISCEX*, 2000.

[27] U. White, G. B.; Pooch. Cooperating security managers: distributed intrusion detection systems. In *Computers & Security, Vol. 15, No. 5*, pages 441–450, 1996.

[28] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *In Proceedings of ACM SIGMETRICS,*, 2003.

[29] S. Zanero. Behavioral Intrusion Detection. In *ISCIS 2004.*

Notes

[1] Moore et al. [16] have shown that it is hard to contain epidemic worms such as Code Red using IP blacklists. In the 150 million alerts we observed in our ISP dataset logs from 40 IDS, only 4% are caused by epidemic worms. Thus, IP blacklisting continues to be an important tool for warding off attack.

[2] The cross correlation is defined as:

$$r_{xy} = \frac{\sum_i (x(i) - \bar{x})(y(i) - \bar{y})}{\sqrt{\sum_i (x(i) - \bar{x})^2}\sqrt{\sum_i (y(i) - \bar{y})^2}}$$

where r_{xy} is the cross correlation, x and y are vectors of equal length, and $\bar{x}$ and $\bar{y}$ are the corresponding means.

[3] Message formatting and exchange protocols, though necessary for IDS collaboration, are beyond the scope of this paper. Some of the existing literature addresses these issues [7, 5, 24].

Perils of Transitive Trust in the Domain Name System

Venugopalan Ramasubramanian and Emin Gün Sirer
Dept. of Computer Science, Cornell University, Ithaca, NY 14853
{ramasv, egs}@cs.cornell.edu

Abstract

The Domain Name System, DNS, is based on nameserver delegations, which introduce complex and subtle dependencies between names and nameservers. In this paper, we present results from a large scale survey of DNS, and show that these dependencies lead to a highly insecure naming system. We report specifically on three aspects of DNS security: the properties of the DNS trusted computing base, the extent and impact of existing vulnerabilities in the DNS infrastructure, and the ease with which attacks against DNS can be launched. The survey shows that a typical name depends on 46 servers on average, whose compromise can lead to domain hijacks, while names belonging to some countries depend on a few hundred servers. An attacker exploiting well-documented vulnerabilities in DNS nameservers can hijack more than 30% of the names appearing in the Yahoo and DMOZ.org directories. And certain nameservers, especially in educational institutions, control as much as 10% of the namespace.

1 Introduction

The Domain Name System (DNS), which resolves host names to IP addresses, is critical to the integrity of Internet services and applications. Yet, the design of DNS poses security risks that are difficult to anticipate and control. DNS relies on a delegation based architecture, where resolution of a name to its IP address requires resolving the names of the servers responsible for that name. Resolving these server names, in turn, depends on additional name resolutions, creating complex interdependencies among DNS servers. Overall, the resolution of a single name is directly or indirectly affected by several servers, and compromise of any of them can severely affect the integrity of DNS and the applications that rely on it.

This paper studies the risks posed by the delegation based architecture for DNS name resolution. Our study, based on a large-scale survey of half a million domain names, answers some of the basic questions about DNS security: How many servers are involved in the resolution of a typical domain name? How easy is it to hijack domains by exploiting well known security holes in DNS servers? Which servers control the largest number of domain names, and how vulnerable are they?

Our survey exposes several new and surprising vulnerabilities in DNS. First, we find that the resolution of a domain name depends on a large trusted computing base of 46 servers on average, not including the root servers. Of this, only 2.2 servers on average are directly designated by the nameowner; the remainder is outside the control of the nameowner. Second, 30% of domain names can be hijacked by compromising just two servers each, where both servers contain publicly-known security loopholes. Finally, about 125 servers control a disproportionate 10% of the namespace. Surprisingly, 25 of these critical servers are operated by educational institutions, which may not have adequate incentives and resources to enforce integrity.

Overall, this study shows that DNS has complex dependencies, where a vulnerability in an obscure DNS server may have far reaching consequences. For example, the domain *fbi.gov* indirectly depends on a server belonging to *telemail.net*, which is vulnerable to four well-known exploits. A malicious agent can easily compromise that server, use it to hijack additional domains, and ultimately take control of FBI's namespace.[1]

The primary contribution of this paper is to expose the inherent risks involved in a basic Internet service. These risks create an artificial dilemma between failure resilience, which argues for more geographically distributed nameservers, and security, which argues for fewer centralized trusted nodes. Our study indicates that many network administrators may not be aware of this artificial tradeoff caused by the current design of DNS, and thus make an uninformed choice between failure resilience and security.

The rest of the paper is organized as follows. The next section provides some background on the delegation based architecture of DNS. Section 3 presents the findings of our survey, and Section 4 summarizes other related DNS surveys. Finally, Section 5 discusses the impact of our findings and concludes.

2 DNS Overview and Threats

DNS namespace is hierarchically partitioned into non-overlapping regions called *domains*. For example, *cs.cornell.edu* is a sub-domain of *cornell.edu*, which in turn is a sub-domain of the top-level domain *edu*, which is under the global root domain. Names within a domain

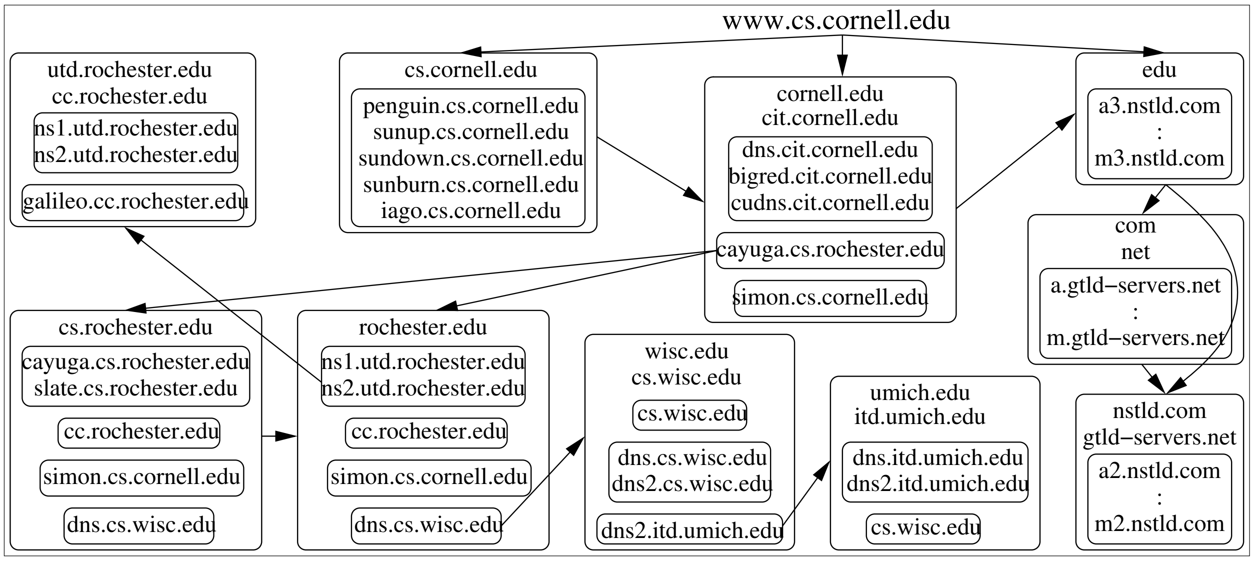


Figure 1: **Delegation Graph: DNS exhibits complex inter-dependencies among nameservers due to its delegation based architecture. For example, the domain name *www.cs.cornell.edu* depends indirectly on a nameserver in *umich.edu*. Arrows in the figure indicate dependencies. Self-loops and redundant dependencies have been omitted for clarity.**

are served by a set of nodes called the *authoritative nameservers* for that domain. At the top of the DNS hierarchy are *root nameservers* and the authoritative nameservers for *top-level domains* (TLDs). The top-level domain namespace consists of generic TLDs (gTLD), such as *.com*, *.edu*, and *.net*, and country-code TLDs (ccTLD), such as *.uk*, *.tr*, and *.in*.

DNS uses a delegation based architecture for name resolution [6, 7]. Clients resolve names by following a chain of authoritative nameservers, starting from the root, followed by the TLD nameservers, down to the nameservers of the queried name. For example, the name *www.cs.cornell.edu* is resolved by following the authoritative namesevers of the parent domains *edu*, *cornell.edu*, and *cs.cornell.edu*. Following the chain of delegations requires additional name resolutions to be performed in order to obtain the addresses of intermediate nameservers.^{2} Each additional name resolution, in turn, depends on a chain of delegations. Overall, these delegations induce complex non-obvious dependencies among nameservers, and can cause unexpected nodes to exert great control over remote domains.

A domain name is said to *depend* on a nameserver if the nameserver could be involved in the resolution of that name. Similarly, a nameserver is said to *affect* a name if the name can involve that nameserver in its resolution. We represent the dependencies among nameservers that directly or indirectly affect a domain name as a *delegation graph*. The delegation graph consists of the transitive closure of all nameservers involved in the resolution of a given name. The nameservers in the delegation graph of a domain name form the *trusted computing base* (TCB) of that name.

Figure 1 illustrates the delegation interdependencies for the name *www.cs.cornell.edu*. In addition to the top-level domain nameservers, the resolution of this name depends on twenty other nameservers, of which only nine belong to the *cornell.edu* domain. Several nameservers that are outside the administrative domain of Cornell have indirect control over Cornell's namespace. In this case, *cornell.edu* depends on *rochester.edu*, which depends on *wisc.edu*, which in turn depends on *umich.edu*. While Cornell directly trusts *cayuga.cs.rochester.edu* to serve its namespace, it has no control over the nameservers that *rochester.edu* trusts.

Compromise of any nameserver in the delegation graph of a name can lead to a hijack of that name. The compromised nameserver can divert DNS requests to malicious nameservers, which effects the hijack by providing false IP addresses; clients can thus be misdirected to servers controlled by attackers and become easy victims of phishing attacks. Surely, it is not the case that all of the nameservers in the delegation graph are involved in every resolution of the name. We distinguish between a *partial hijack*, where an attacker compromises a few nameservers and diverts some queries for the targeted name, and a *complete hijack*, where an attacker compromises enough nameservers to guarantee the misdirection of all the queries for the name.

Attackers can use a combination of techniques, including systematic break-ins and denial of service attacks, to disrupt nameservice. Commonly used nameserver softwares, such as BIND, have well-documented security loopholes, which can be exploited using standard crack tools to break into the vulnerable nameservers [3]. Targeted denial of service attacks through link saturation and overloading on some nameservers further exacerbates the impact of the break-ins by increasing the number of requests passing through the exploited nameservers. Overall, the dele-

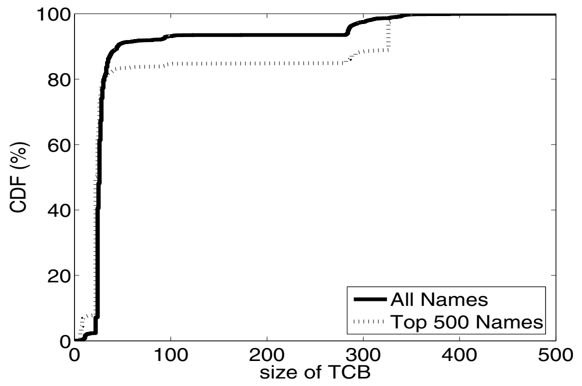


Figure 2: **Size of TCB: DNS Name resolution depends on a large number of nameservers. On average, name resolution involves 46 nameservers, while a sizable fraction of names depend on more than 100 nameservers.**

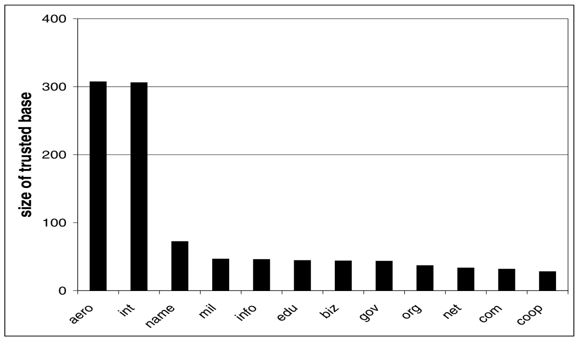


Figure 3: **Average TCB Size for gTLD Names: Names in *.aero* and *.int* have significantly larger TCBs.**

gation graph facilitates attackers to carefully select targets that maximize the impact of attacks and to take over large portions of the namespace.

3 Survey Results

We performed a large-scale survey to understand the risks posed by DNS delegations. We collected 593160 unique webserver names by crawling the Yahoo! and DMOZ.org directories. These names are distributed among 196 distinct top-level domains. Since the names were extracted from Web directories, these names are representative of the sites people actually care about. We then queried DNS for these names and recorded the chain of nameservers that were involved in their resolution. Totally, 166771 nameservers were discovered in this process. We thus obtained a snapshot of the DNS dependencies as it existed on July 22, 2004.

We study three different aspects of the dependencies to quantify the security risks in DNS. First, we examine the size of the trusted computing base for each name to determine which names are most vulnerable. Second, we study how software loopholes in DNS servers can be exploited to hijack domain names. Finally, we determine the most valuable nameservers, which affect large portions of the namespace, and explore how securely they are managed.

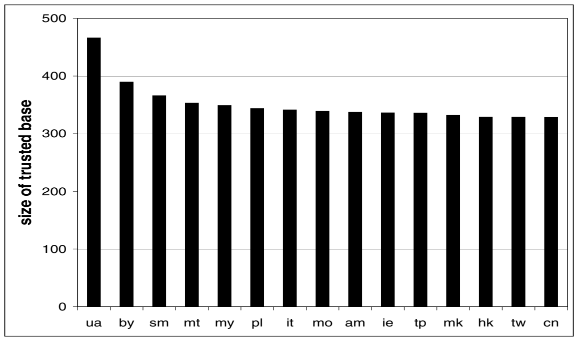


Figure 4: **Average TCB Size for ccTLD Names: Some ccTLDs rely on, and are vulnerable to compromises in, a large number of servers.**

3.1 Most Vulnerable Names

The vulnerability of a DNS name is tied to the number of servers in its trusted computing base, whose compromise could potentially misdirect clients seeking to contact that server. Larger TCBs provide attackers with a wider choice of targets to attack. Further, larger TCBs also imply more complex and deeper dependencies among nameservers making it more difficult for the nameowner to control the integrity of the servers it depends on. In this section, we characterize the TCB size of the surveyed names.

Figure 2 plots the cumulative distribution of TCB sizes not including the root nameservers, which belong to the TCBs of all the domain names. Our survey shows that TCB size follows a heavy-tailed distribution with a median of 26 nameservers, and an average of 46 nameservers; about 6.5% of the names has a TCB of greater than 200 nameservers. We computed the TCB by counting the number of distinct server names in the delegation graph. Since distinct names referring to the same machine may cause the TCB to appear larger, we also computed the number of distinct IP addresses in the delegation graphs. TCB size based on IP addresses has the same median (26), while the average decreases marginally to 44.

One might expect that the administrators of the popular websites would be better aware of the security risks and keep their DNS dependencies small. To test this hypothesis, we separately studied the TCB sizes for the 500 most popular websites reported by *alexa.org*. Figure 2 shows that these names are more vulnerable; they depend on 69 nameservers on average, and 15% of them depend on more than 200 nameservers.

Next, we study the TCB sizes for names belonging to different TLDs. Figures 3 and 4 plot in decreasing order the TCB sizes for names in the generic TLDs, and the fifteen most vulnerable country-code TLDs, respectively. Overall, ccTLD names have a much higher average TCB size of 209 nameservers than gTLD names, whose average is 87 nameservers. GTLDs *aero* and *int* have considerably larger TCBs than other gTLDs, and among the ccTLDs Ukraine, Belarus, San Marino, Malta, Malaysia, Poland and Italy, in that order, are the most vulnerable.

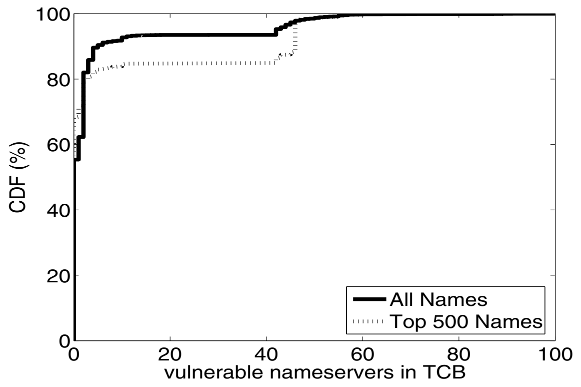


Figure 5: **Vulnerable Nameservers in TCB: 45% of the names depend on at least one nameserverver with known vulnerability.**

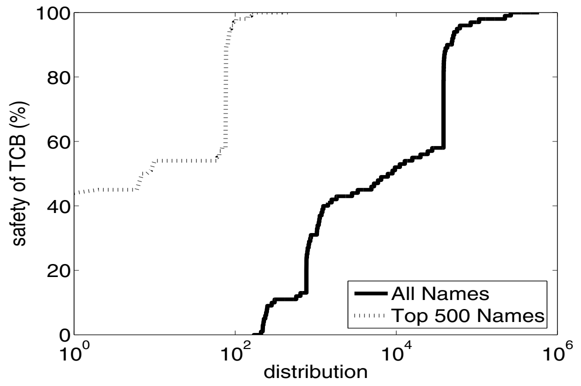


Figure 6: **Percentage of Non-Vulnerable Nodes in TCB: A few names have their entire TCB vulnerable to known exploits.**

We examined the dependencies to determine why certain domain names (e.g., names in *aero* and *int*) have much larger TCBs than others. We find that names with larger TCBs typically have authoritative nameservers distributed across distant domains. Improving availability in the presence of network outages is one of the primary reasons why administrators delegate to, and implicitly trust, nameservers outside their control. Extending trust to a small number of nameservers that are geographically distributed may provide high resilience against failures. However, DNS forces them to trust the entire transitive closure of the all names that appear in the physical delegation chains.

Sometimes even top-level domains are set up such that it is impossible to own a name in that subdomain and not depend on hundreds of nameservers. Ukrainian names seem to suffer from many such dependencies. The most vulnerable name in our survey, *www.rkc.lviv.ua*, depends on nameservers in the US including Berkeley, NYU, UCLA, as well as many locations spanning the globe: Russia, Poland, Sweden, Norway, Germany, Austria, France, England, Canada, Israel, and Australia.[3] It is likely that the Ukrainian authorities do not realize their dependency on servers outside their control. A cracker that controls a nameserver at Monash University in Australia can end up hijacking the website of Ukrainian government. DNS creates a small world after all!

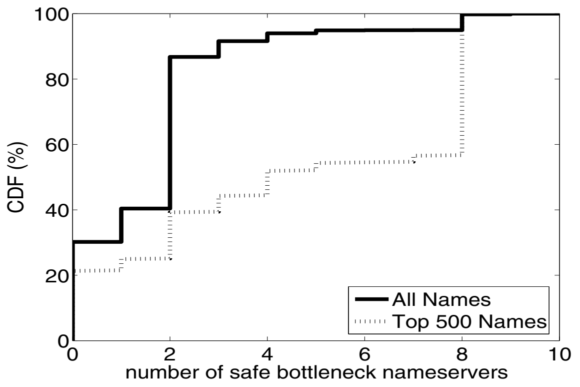


Figure 7: **DNS Nameserver Bottlenecks: 30% percentage of names can be completely hijacked by compromising a critical set of vulnerable bottleneck nameservers.**

3.2 Impact of Known Exploits

As part of our survey, we also collected version information for nameservers using BIND, the most widely-used DNS server, where possible. Different versions of BIND contain well-documented software bugs [3]. We combine known vulnerabilities with the delegation graphs of domain names to explore which names are easily subjected to compromise. For nameservers whose vulnerabilities we do not know, we simply assume that they are *non-vulnerable*; hence, the results presented here are optimistic.

Of the 166771 nameservers we surveyed, 27141 have known vulnerabilities. A naive expectation might be that, with 17% vulnerable nameservers, only 17% of the names would be affected. Instead, these vulnerabilities affect 264599 names, approximately 45%, because transitive trust relationships “poison” every path that passes through an insecure nameserver.

For example, *www.fbi.gov* is vulnerable to being hijacked, along with all other names in the *fbi.gov* domain. The *fbi.gov* domain is served by two machines named *dns.sprintip.com* and *dns2.sprintip.com*. The *sprintip.com* domain is in turn served by three machines named *reston-ns[123].telemail.net*. Of these machines, *reston-ns2.telemail.net* is running an old nameserver (BIND 8.2.4), with four different known exploits against it (libbind, negcache, sigrec, and DoS_multi) [3]. Having compromised *reston-ns2*, an attacker can divert a query for *dns.sprintip.com* to a malicious nameserver, which can then divert queries for *www.fbi.gov* to any other address, hijacking the FBI’s website and services.

Figure 5 shows the cumulative distribution of the number of vulnerable nameservers in the TCBs of surveyed names. 45% of DNS names depend on at least one vulnerable nameserver, and can be compromised by launching well-known, scripted attacks. Figure 6 shows the percentage of nodes with no known bugs in the TCBs of surveyed names. Surprisingly, a few names do not have any non-vulnerable nameservers in their TCB; these names belong to the ccTLD *ws*, which relies on older buggy versions of

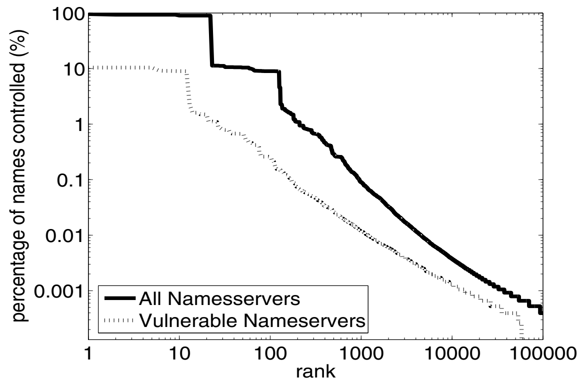


Figure 8: **Percentage of Names Controlled by Nameservers: Some nameservers with known vulnerabilities affect a large percentage of names.**

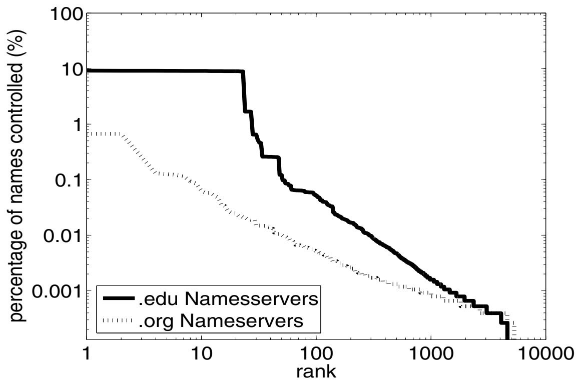


Figure 9: **Percentage of Names Controlled by Nameservers in *.edu* and *.org* Domains: Some nameservers in educational institutions and non-profit organizations affect large percentage of names.**

BIND. Overall, the average number of vulnerable servers is 4.1, about 9% of the average TCB size. The extent of vulnerability in the TCBs of the 500 most popular names is also high (7.6), about 11% of the average TCB size.

We examined the chances of a complete domain hijack by counting the minimum number of nameservers that need to be attacked in order to completely take over a domain. Such critical bottleneck nameservers can be determined by computing a min-cut of the delegation graph. Figure 7 shows the number of non-vulnerable nameservers in the min-cut of the delegation graphs.

Surprisingly, about 30% of domain names have a min-cut consisting entirely of vulnerable nameservers. The average size of a min-cut is 2.5 nameservers. This implies that these domain names can be completely hijacked by compromising less than three machines on average. Moreover, another 10% of domain names have only one non-vulnerable nameserver in their min-cut. A denial of service attack on the non-vulnerable nameserver, coupled with the compromise of the other vulnerable bottleneck nameservers, is sufficient to completely hijack these domains.

3.3 Most Valuable Nameservers

The value of a DNS nameserver is tied to the role it plays in name resolution. We model the value of a nameserver as being proportional to the number of domain names which depend on that nameserver. It is these high profile servers whose compromise would put the largest portions of the DNS namespace in jeopardy. Attackers are likely to focus their energies on such high-leverage servers; if the effort to break into a vulnerable nameserver is constant, then breaking into a nameserver that affects a large number of names provides a higher payoff.

Figure 8 shows the percentage of names affected by nameservers, ranked in the order of importance. It also gives a distribution of names affected by nameservers with known exploits. An average nameserver is involved in the resolution of 166 externally visible names, and the median is 4. This is the number of externally visible names that appear in well-known Web directories, and does not include automatically generated DHCP names or other DNS names that receive few, if any, lookups.

While an attacker targeting random nameservers would likely compromise only a few sites, a little bit of targeting can yield nameservers with great leverage. Figure 8 shows that about 125 nameservers each affect more than 10% of the surveyed names. Of these high profile nameservers, only about 30 are well-maintained gTLD nameservers. Several vulnerable nameservers affect large portions of the namespace; about 12 of the 125 high profile nameservers have well-known loopholes.

There are many valuable nameservers operated by institutions that may not be equipped to or willing to take on the DNS task. Figure 9 shows a distribution of names served by machines belonging to the *.edu* and *.org* domains. These nameservers are operated by entities such as universities, non-profit organizations, and so forth, whose primary business is not to provide networking services. These institutions, unlike ISPs, typically do not have a financial relationship with the owners of the names they serve, and thus lack the fiduciary incentives for providing correct, secure service that an ISP has. These institutions take on an additional risk by placing their servers at critical locations in the DNS hierarchy; they may be liable if their servers are taken over and used to hijack a DNS domain.

4 Related Work

Several surveys and measurement studies have been performed on DNS. However, they have typically focused on the performance and availability of DNS.

In 1988, Mockapetris and Dunlap published a retrospective study on the development of DNS identifying its successful features and shortcomings [8]. Several measurement studies since then have provided insights into the performance of the system. A detailed study of the effectiveness of caching on lookup performance is presented by Jung et al. in [4, 5]. Park et al. [10] explore the different causes for performance delays seen by DNS clients.

Huitema and Weerahandi [2] and Wills and Shang [14] study the impact of DNS delays on Web downloads. The impact of server selection on DNS delays is measured by Shaikh et al. [12].

Two recent surveys by Pappas et al. [9] and Ramasubramanian and Sirer [11] focus on availability limitations of DNS stemming from its hierarchical structure. These studies show that most domain names are served by a small number of nameservers, whose failure or compromise prevents resolution of the names they affect.

This paper studies a fundamentally different, yet crucial, aspect of DNS design: the security vulnerabilities that stem from the delegation based architecture of DNS. It exposes the risks posed by non-obvious dependencies among DNS servers, and highlights the tradeoff between availability and security.

5 Discussion and Summary

DNS is a complex system, where a vulnerability in an obscure nameserver can have far-reaching consequences, and trust relationships are hard to specify and bound. Even if the name owners are diligent and check the extent of dependencies at the time of name creation, trust relationships can change undetected.

The main culprit here is the reliance on transitive trust [13]. Nameserver delegations induce a dependency graph, and concerns, including failure resilience and independent administration, enable the resulting dependency graphs to grow large and change dynamically. It is a well-accepted axiom of computer security that a small trusted computing base is highly desirable, since smaller TCBs are easier to secure, audit and manage. Our survey finds that the TCB in DNS is large and can include more than 400 nodes. An average name depends on 46 nameservers, while the average in some top-level domains exceeds 200.

This study shows that one in three Internet names can be hijacked using publicly-known exploits. This points to the Domain Name System as a significant common vulnerability. It is highly unlikely that an attacker can break into a third of the webservers around the globe; firewalls, hardened kernels, and intrusion detection tools deter direct attacks on webservers. But DNS enables attackers to hijack one in three sites, thus gaining the ability to masquerade as the original site, obtain access to their clients, potentially collect passwords, and possibly spread misinformation. High-profile domains, including those belonging to the FBI and many popular sites, are vulnerable because of problems stemming from the way DNS uses delegations.

A better approach is required to achieve name security on the Internet. Deployment of DNSSEC [1] can help, but DNSSEC continues to rely on the same physical delegation chains as DNS during lookups. Complex dependencies in name resolution means that a much wider acceptance of DNSSEC is required for it to be effective, since every path in the delegation graph needs to be secured. And even if all nameservers support DNSSEC, attackers can exploit vulnerabilities outlined in this paper to launch DoS attacks on Web services and disrupt name resolution. As a stopgap measure, network administrators have to be aware of the vulnerabilities in DNS and be more diligent about where they place their trust.

Notes

[1] We reported this vulnerability to the Department of Homeland Security and the servers have since been upgraded; we do not know if the vulnerability has been fix ed.

[2] While DNS uses glue records, which provide cached IP addresses for nameservers, as an optimization, glue records are not authoritative.

[3] A complete list of nameservers this name depends on can be found in http://www.cs.cornell.edu/people/egs/beehive/dnssurvey.html. We maintain an active website listing the results of the survey presented here.

References

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the Domain Name System Security Extensions. *Request for Comments 4035*, Mar. 2005.

[2] C. Huitema and S. Weerahandi. Internet Measurements: The Rising Tide and the DNS Snag. In *Proc. of ITC Specialist Seminar on Internet Traffic Measurement and Modeling*, Monterey, CA, 2000.

[3] Internet Systems Consortium. BIND Vulnerabilities. http://www.isc.org/sw/bind/bind-security.php, Feb. 2004.

[4] J. Jung, A. Berger, and H. Balakrishnan. Modeling TTL-based Internet Caches. In *Proc. of IEEE International Conference on Computer Communications*, San Francisco, CA, Mar. 2003.

[5] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and Effectiveness of Caching. In *Proc. of SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov. 2001.

[6] P. Mockapetris. Domain Names: Concepts and Facilities. *Request for Comments 1034*, Nov. 1987.

[7] P. Mockapetris. Domain Names: Implementation and Specification. *Request for Comments 1035*, Nov. 1987.

[8] P. Mockapetris and K. Dunlop. Development of the Domain Name System. In *Proc. of ACM SIGCOMM*, Stanford, CA, 1988.

[9] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *Proc. of ACM SIGCOMM*, Portland, OR, Aug. 2004.

[10] K. Park, V. Pai, and L. Peterson. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proc. of Symposium on Operating Systems Design and Implementation*, 2004.

[11] V. Ramasubramanian and E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proc. of ACM SIGCOMM*, Portland, OR, Aug. 2004.

[12] A. Shaikh, R. Tewari, and M. Agarwal. On the Effectiveness of DNS-based Server Selection. In *Proc. of IEEE International Conference on Computer Communications*, Anchorage, AK, Apr. 2001.

[13] K. Thompson. Reflections on Trusting Trust. *Comm. of the ACM*, 27(8), Aug. 1984.

[14] C. E. Wills and H. Shang. The Contribution of DNS Lookup Costs to Web Object Retrieval. Technical Report TR-00-12, Worcester Polytechnic Institute, July 2000.

Flooding Attacks by Exploiting Persistent Forwarding Loops

Jianhong Xia, Lixin Gao, Teng Fei
University of Massachusetts at Amherst
{jxia, lgao, tfei}@ecs.umass.edu

ABSTRACT

In this paper, we present flooding attacks that exploit routing anomalies in the Internet. In particular, we focus on routing anomalies introduced by persistent forwarding loops. Persistent forwarding loops may share one or more links with forwarding paths to reachable addresses. An attacker can exploit persistent forwarding loops to overload the shared links to disrupt the Internet connectivity to those reachable addresses.

To understand the extent of this vulnerability, we perform extensive measurements to systematically study persistent forwarding loops and the number of network addresses that can be affected. We find that persistent forwarding loops do exist in the current Internet. About 0.2% of routable addresses experience persistent forwarding loops and 0.21% of routable addresses can be attacked by exploiting persistent forwarding loops. In addition, 85.16% of the persistent forwarding loops appear within destination domains and they can be observed from various locations, which makes it possible to launch attacks from many vantage points. We also find that most persistent forwarding loops are just two hops long, which enables an attacker to amplify traffic to persistent forwarding loops significantly. To the best of our knowledge, this is the first study of exploiting the vulnerability of persistent forwarding loops to launch DDoS attacks.

1 INTRODUCTION

Distributed denial of service (DDoS) attack is one of the most prevailing threats in the Internet. In general, DDoS attacks send traffic from a large number of compromised hosts to deplete network or host resources needed by the victim. In this paper, we present flooding attacks that exploit routing anomalies in the Internet. In particular, we focus on a critical vulnerability in network routing architecture that is caused by persistent forwarding loops. Forwarding loops have been observed in previous measurement studies [3, 8, 11, 12]. Although transient forwarding loops disappear after routing protocol convergence, forwarding loops caused by configuration errors can last for a long time. In addition to obvious issues that persistent forwarding loops can blackhole network addresses, they can also be exploited to overload links that appear in the persistent forwarding loops.

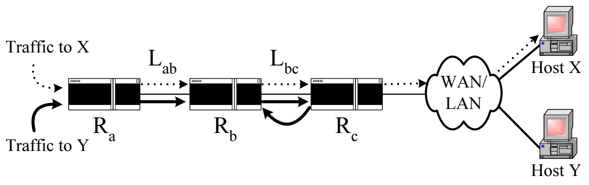


Figure 1: Flooding Attacks by Exploiting Persistent Forwarding Loops

Fig. 1 shows an example of a flooding attack that exploits persistent forwarding loops. Traffic to host X traverses routers R_a, R_b, R_c and other network devices to reach host X. At the same time, traffic to host Y also traverses routers R_a, R_b and R_c. However, due to misconfigurations in router R_c, traffic to host Y will be forwarded back to R_b. Therefore, any packet destined to Y falls into the loop between R_b and R_c, and will be dropped until its time-to-live (TTL) expires. In this scenario, link L_{bc} can be flooded if malicious attackers deliberately send a large amount of traffic to host Y. Host X would experience denial of service. Since traffic traversing a persistent forwarding loop typically traverses the links in the loop several times before being dropped, attackers take much less effort to launch flooding attacks, and therefore making the attacks stealthy. Since network operators can see high congestion only on the shared link L_{bc} but not on other links such as L_{ab}, without packet or flow-level measurements on the shared link, this kind of attack is hard to detect.

We perform extensive measurements to systematically study persistent forwarding loops. We find that persistent forwarding loops do exist in the current Internet. About 0.2% of routable addresses experience persistent forwarding loops and 0.21% of routable addresses can be attacked by exploiting persistent forwarding loops. In addition, 85.16% of the persistent forwarding loops appear within destination domains and they can be observed from various locations, which makes it possible to launch attacks from many vantage points. We also find that most persistent forwarding loops are just two hops long, which enables an attacker to amplify traffic to persistent forwarding loops significantly.

The remainder of this paper is structured as follows. Section 2 introduces the concept of persistent forwarding loops. Section 3 describes measurement design and data collection. Section 4 characterizes persistent forwarding loops. Section 5 exploits flooding attacks using persistent forwarding loops. We summarize the paper in Section 6.

2 PERSISTENT FORWARDING LOOPS

2.1 Concept of Forwarding Loops

In general, a packet from source s traverses a sequence of routers to reach destination d. A packet experiences a *forwarding loop* if it traverses a set of routers more than once. One of the powerful tools for discovering the forwarding path from a source to a destination is traceroute, which returns a sequence of router interfaces on the forwarding path. We denote the sequence of router interfaces in a trace from s to d as $(r_1, r_2, \ldots, r_n)$. If $r_i = r_j$ and $i \neq j$, then the trace contains a forwarding loop $(r_i, \ldots, r_j)$. The *length of forwarding loop* $(r_i, \ldots, r_j)$ is $j - i$.

2.2 Transient and Persistent Forwarding Loops

Forwarding loops can be transient or persistent. *Transient forwarding loops* are the forwarding loops that resolve themselves without human intervention or network topology changes. They may occur during routing protocol convergence [1]. Hengartner et al. [3] has demonstrated that forwarding loops exist in the Sprint network by analyzing packet traces. In general, forwarding loops are transient if they will disappear once the routing protocol converges. However, some forwarding loops will not disappear without human intervention or network topology changes. We refer to those forwarding loops as *persistent forwarding loops*.

2.3 Shadowed Address and Imperiled Address

If there is a persistent forwarding loop from source s to destination d, we refer to the IP address of d as a *shadowed address*. For example, the IP address of host Y in Fig. 1 is a shadowed address. Note that the links in persistent

Hop	Traceroute to shadowed address 81.181.31.127	Traceroute to imperiled address 80.96.192.10
1	128.119.91.254	128.119.91.254
2	128.119.2.238	128.119.2.238
...	...	...
18	166.49.147.134	166.49.147.134
19	195.39.208.82	195.39.208.66
20	193.226.179.18	193.226.179.18
21	**193.226.130.226**	**193.226.130.226**
22	**194.176.189.42**	**194.176.189.42**
23	**193.226.130.226**	194.105.11.178
24	**194.176.189.42**	80.96.192.10
25	**193.226.130.226**	
26	**194.176.189.42**	
27	**193.226.130.226**	
28	**194.176.189.42**	
29	**193.226.130.226**	
30	**194.176.189.42**	

Figure 2: A Shadowed Address and an Imperiled Address

forwarding loops may still be able to carry traffic to other reachable addresses. That is, a persistent forwarding loop may share one or more links with forwarding paths to the IP addresses other than shadowed addresses. If a destination d' is reachable and the forwarding path to d' shares one or more links with a persistent forwarding loop, we refer to the IP address of d' as an *imperiled address*. For example, the IP address of host X in Fig. 1 is an imperiled address. Imperiled address is named so because the address is in a dangerous situation and it may suffer from the potential threats posed by the persistent forwarding loops.

An example of a shadowed address and an imperiled address is shown in Fig. 2. In this example, traffic to the shadowed address *81.181.31.127* falls into a persistent forwarding loop between routers *193.226.130.226* and *194.176.189.42*. However, traffic to the imperiled address *80.96.192.10* relies on the same link that appears in the forwarding loop to reach the destination.

3 DATA COLLECTION

3.1 Measurement Design

We use traceroute to discover forwarding paths in our study. Our goal is to identify all possible persistent forwarding loops in the Internet. In order to reduce the measurement overhead, we select a set of representative IP addresses to trace. We intend to select as few IP addresses as we can, while trying to discover as many forwarding paths as possible. Most networks or their subnets are allocated a set of contiguous IP address blocks, and forwarding decision in a router is based on the destination IP address only. Therefore, tracing different IP addresses in the same contiguous IP block from a source may observe the same for-

Table 1: Summary on Measurement Design and Trace Data

Data Set	Fine-grained Prefix Selection	# of Selected IP Addresses/Prefix	# of Traces for Each Selected IP Address	# of Prefixes Traced	# of IP Addresses Traced	# of Traces Collected
D_A	all fine-grained prefixes	2	once	5,238,191	9,259,257	9,384,350
D_B	10% of candidate prefixes	2	5∼7 times	12,983	21,212	139,739
D_C	35% of shadowed prefixes	50	once	3,705	171,218	187,980
D_{D1}	46% of shadowed prefixes	4	twice on average	4,894	19,762	41,556
D_{D2}	46% of shadowed prefixes	4	twice on average	4,894	20,691	44,969
D_{D3}	46% of shadowed prefixes	4	twice on average	4,894	20,657	44,936
D_{D4}	46% of shadowed prefixes	4	twice on average	4,894	17,825	34,873

warding path. Typically, a /24 block is used as the smallest unit, thus the forwarding path to any IP address in a /24 block should represent forwarding paths to all IP addresses in that block. Therefore, we design our measurement by selecting a couple of IP addresses in each /24 block to perform traceroute.

It should be noted that not all IP addresses have been allocated. Even we can obtain a list of all allocated IP addresses [5], not all allocated IP addresses have been used in the current Internet. To reduce the overhead of our measurement, we select the set of IP addresses based on the information from BGP routing tables in the Route-Views Project [10]. The RouteViews server peers with BGP routers in many large ISPs such as AT&T and Sprint. We refer to the prefixes in the BGP routing tables in the RouteViews server as *routable prefixes*, and refer to the IP addresses covered by routable prefixes as *routable addresses*.

We divide all routable prefixes whose lengths are shorter than 24 into multiple /24 prefixes, and keep routable prefixes whose lengths are no shorter than 24 unchanged. For example, prefix 12.0.0.0/8 is divided into $65,536$ prefixes represented by 12.x.x.0/24. All prefixes in our measurement have a length of at least 24. We refer to these prefixes as *fine-grained prefixes*.

Since the forwarding paths to only a limited number of IP addresses are used to represent forwarding paths to all IP addresses in a fine-grained prefix, we extend the concept of persistent forwarding loops to fine-grained prefixes. We say that there is a persistent forwarding loop to a fine-grained prefix p from source s if we find a destination d_p in p that experiences persistent forwarding loops from s. Similarly, if a fine-grained prefix p contains a shadowed address, we refer to prefix p as a *shadowed prefix*. If a fine-grained prefix q contains an imperiled address, we refer to prefix q as an *imperiled prefix*.

3.2 Data Sets

We have collected four sets of trace data in this study. Most traces are collected from one location. We also collect additional traces on different hosts from various locations to support our findings. Unless otherwise specified, all traces are collected from the campus network of University of Massachusetts at Amherst.

The first data set, D_A, is for detecting forwarding loops in all fine-grained prefixes. From a total of 0.17 million routable prefixes, we obtain about 5.36 million fine-grained prefixes. Due to security and privacy concerns posed by networks owned by governmental and military agencies, we filter out their prefixes according to WHOIS [4]. After filtering, 5.24 million fine-grained prefixes are traced. To reduce the overhead of our measurement, we perform traceroute to two IP addresses in each prefix, the first one and a random one. We collect 9.38 million traces in D_A by tracing to 9.26 million IP addresses. We refer to those fine-grained prefixes with forwarding loops in D_A as *candidate prefixes*. Since some selected IP addresses are traced twice during the experiment, the number of traces is slightly greater than the number of IP addresses in D_A.

The second data set, D_B, is for detecting persistent forwarding loops. To identify persistent forwarding loops, we trace to candidate prefixes and perform traceroute multiple times. Although we can observe forwarding loops from a single trace, it is impractical to monitor the network forever to identify persistent forwarding loops. Thus, we adopt an approximate criterion with respect to the general time scale of routing convergence. We trace an IP address d multiple times within four days. If there is a forwarding loop for all traces to d, we classify the forwarding loop as a persistent forwarding loop. In this case, d is a shadowed address and the fine-grained prefix that contains d is a shadowed prefix. To collect D_B, we trace to 10% of candidate prefixes and select two IP addresses in each prefix, a first one and a random one. Each selected IP address is traced at least 5 up to 7 times. We collect $139,739$ traces by tracing to $21,212$ IP addresses. Since in some prefixes only one IP address is traced during the experiment, the number of selected IP addresses is less than twice of number of prefixes in D_B.

After we identify persistent forwarding loops and shadowed prefixes from D_B, we further examine the forwarding consistency to multiple IP addresses in shadowed prefixes, and the observability of persistent forwarding loops from different locations. For these purposes, we collect two additional data sets, D_C and D_D. In D_C, we trace to about 35% of the shadowed prefixes and select 50 random IP addresses in each prefix. We collect data D_D from four hosts in PlanetLab [9] that are located in Asia, Europe, US east coast and US west coast. We denote data sets from four hosts as D_{D1}, D_{D2}, D_{D3}, and D_{D4} respectively. For each of them, we trace to about 46% of shadowed prefixes and select 4 random IP addresses in each prefix. Table 1 summarizes our measurement design and data sets.

4 CHARACTERIZING PERSISTENT FORWARDING LOOPS

A trace of traceroute normally contains a sequence of router interface addresses. However, some traces may contain "*" or "!" when routers do not send back ICMP packets, replies get lost or filtered, or destinations cannot be reached. To reduce ambiguity, we filter out the traces that contain "*" or "!" between two appearances of a same address. We also filter out the traces where the same address appears continuously because forwarding loops could not be constructed by a single router interface.

4.1 Prevalence of Shadowed Prefixes and Imperiled Prefixes

4.1.1 Shadowed Prefixes

In our measurement, we identify the candidate prefixes from D_A and perform traceroute to these candidate prefixes to collect D_B. We then analyze D_B to identify persistent forwarding loops and shadowed prefixes.

Among 5.24 million prefixes traced in D_A, $139,278$ of them are identified as candidate prefixes. If we convert them into IP addresses, they cover about 2.66% of routable IP addresses.

From data D_B that traces to 10% of candidate prefixes, we obtain $9,630$ persistent forwarding loops, and identify $10,569$ prefixes as shadowed prefixes. If we convert shadowed prefixes into IP addresses, we find that 81.39% of IP addresses in our sampled space have persistent forwarding loops. This number constitutes 0.2% of all routable IP addresses. Shadowed prefixes are located in $2,950$ ASes, which suggests that IP addresses experiencing forwarding loops are originated from a large number of ASes. We believe that shadowed addresses in the Internet could be much more than what we have found in D_B because we trace to only 10% of candidate prefixes.

Note that we trace a limited number of IP addresses in each fine-grained prefix to collect forwarding paths. In order to confirm that, not only the selected IP addresses in shadowed prefixes experience forwarding loops, but other IP addresses in shadowed prefixes also experience forwarding loops, we use D_C for this study. 67.96% of shadowed prefixes in D_C confirm that all additional sampled hosts have forwarding loops. We further investigate the reason that not all additional sampled hosts have forwarding loops in shadowed prefixes. We find that 73.41% of them are caused by the fact that infrastructure addresses (deployed for router interfaces) are sampled. For example in Fig. 1, although there is a forwarding loop when tracing to host Y, there is no forwarding loop if we trace to the interface address of R_c. It suggests that multiple IP addresses in the shadowed prefixes experience forwarding loops.

4.1.2 Imperiled Prefixes

As mentioned in Section 2, the vulnerability of persistent forwarding loops does not come from the shadowed addresses themselves. Rather, it comes from the shared links between persistent forwarding loops and the forwarding paths to imperiled addresses. To understand the extent of this vulnerability, we estimate the prevalence of imperiled addresses in the Internet.

The basic idea on identifying imperiled addresses is to find those IP addresses that are reachable and their forwarding paths share one or more links with persistent forwarding loops. It is not easy to fully identify the imperiled addresses in the Internet without a global view of forwarding paths from a source to a destination. In our experiment, we estimate the number of imperiled addresses/prefixes from D_A. Any reachable address in D_A that uses one or more links in a persistent forwarding loop is marked as an imperiled address. The fine-grained prefixes containing any imperiled address are marked as imperiled prefixes. Based on the persistent forwarding loops found in Section 4.1.1 and the traces in D_A, $10,828$ of fine-grained prefixes are identified as imperiled prefixes. If we convert them into IP addresses, about 0.21% of all routable IP addresses are imperiled addresses. These imperiled addresses could be the potential victims when the vulnerability on persistent forwarding loops is exploited. These imperiled prefixes are originated from $1,516$ ASes, so the potential victims widely spread in various domains. We show an imperiled address discovered in our measurement in Fig. 2.

Note that not all persistent forwarding loops share their links with forwarding paths to imperiled prefixes. Among $9,630$ persistent forwarding loops, only 6.33% of them share links with forwarding paths to imperiled addresses. We call those shadowed addresses (prefixes) that can be exploited for attacking imperiled addresses *dark addresses* (*prefixes*). Among $10,569$ shadowed prefixes, only 5.64% of them are dark prefixes. With the growth and evolution of the Internet, some shadowed prefixes may become dark prefixes. Generally, a persistent forwarding loop shares one or more links with forwarding paths to only up to two imperiled prefixes. However, some persistent forwarding loops may share one or more links with forwarding paths to as many as $1,000$ imperiled prefixes. Flooding such shared links can result in denial of service to a large number of imperiled addresses.

4.2 Properties of Persistent Forwarding Loops

4.2.1 Location of Persistent Forwarding Loops

Identifying the location of persistent forwarding loops is helpful for us to understand where they occur. Persistent forwarding loops may occur within the destination domains, or across one or more other domains. It is difficult to accurately map infrastructure IP addresses to AS numbers [7]. However, because the most serious inaccuracies occur at AS boundaries, the accuracy may not be a problem if we only identify persistent forwarding loops that occur within destination domains. We consider that a persistent forwarding loop occurs within the destination domain if all interface addresses that appear in the loop are originated from the same AS as the shadowed address.

Among $91,090$ traces with persistent forwarding loops, 85.16% of them occur in destination domains. It suggests that most persistent forwarding loops are close to the shadowed addresses rather than in the core of Internet. When persistent forwarding loops occur in destination domains, we conjecture that traffic to the shadowed addresses from different locations will most likely fall into these loops although they may traverse different paths in the core of Internet.

To confirm our conjecture, we collect additional traces, D_{D1}, D_{D2}, D_{D3} and D_{D4} on various hosts to verify the observability of persistent forwarding loops from different locations. We find that, persistent forwarding loops to about 90% of shadowed prefixes can still be observed from all four locations. Given that most persistent forwarding loops happen in destination domains, it is not surprising that they can be observed from various locations. However, comparing with the result that 85.16% of traces with persistent forwarding loops occur in destination domains, we conclude that although some persistent forwarding loops may not occur in destination domains, they can still be observed from different locations. It suggests that attackers are able to exploit persistent forwarding loops from different locations, which make this vulnerability more critical.

4.2.2 Length of Persistent Forwarding Loops

The length of persistent forwarding loops is important for us to understand traffic amplification factor in the links that appear in the persistent forwarding loops. When a packet enters a persistent forwarding loop, it may traverse the links in the loop multiple times before its TTL expires. The shorter the length of a persistent forwarding loop is, the more times a packet traverses the links in the loop. We find that, among $9,630$ persistent forwarding loops, over 88.82% of them have a length of 2, which can significantly amplify the amount of traffic to shadowed addresses in the links that appear in the loops. About 8.71% of them have a length of 3 to 10. The rest of them have a length of 11 or longer. Several persistent forwarding loops have a length as long as 20. The distribution of length of persistent forwarding loops is shown in Fig. 3.

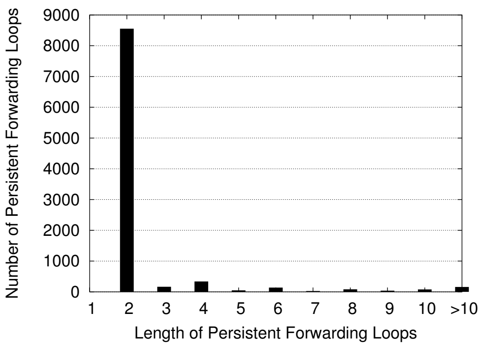


Figure 3: Distribution of Length of Persistent Forwarding Loops

4.3 Possible Causes of Persistent Forwarding Loops

It is hard to identify the root causes of persistent forwarding loops without information about configurations on the involved routers. We conjecture that, the most possible cause of persistent forwarding loops is misconfiguration of the common usages of default routes and static routes. Several examples in [2] have shown that forwarding loops can happen if BGP or static routes are incorrectly configured. BGP misconfigurations are common today in the Internet [6].

To understand how misconfigurations can easily lead to persistent forwarding loops, we show an example that a network administrator neglects to configure a "pull-up route" at a border route to his upstream provider. Provider P owns 18.0.0.0/8 and delegates 18.1.0.0/16 to its customer C. The provider's border router might have a static route directing traffic for 18.1.0.0/16 to the customer's border router. The customer's border router, in turn, might have routes for some subnets of 18.1.0.0/16, such as 18.1.1.0/24 and 18.1.2.0/24, but not for others. The customer's border router may also have a default route (e.g., 0.0.0.0/0) pointing to the link back to the provider's router, for access to the Internet. That would cause a persistent forwarding loop for all traffic destined to addresses in the range of 18.1.3.0 to 18.1.255.255.

In the above case, the forwarding loop is two hops long and near the destination domain. However, when the customer C is multi-homed, the same misconfiguration may also lead to a persistent forwarding loop that occurs across multiple domains. For example, if the customer C has another provider B and prefers to use its link to provider B for outbound traffic. Therefore, the customer C's border router has a default route 0.0.0.0/0 to provider B. The customer C also prefers to use the link from provider P for inbound traffic. In this case, when the customer C receives any traffic destined to addresses in the range of 18.1.3.0 to 18.1.255.255, it will forward the traffic to its provider B. The provider B will forward the traffic to its own provider or neighbors. Eventually the traffic will return to the provider P and reach the customer C again. That would cause a persistent forwarding loop occurring across multiple domains and has a loop length longer than 2.

To prevent this, the customer needs to configure a "null route" for 18.1.0.0/16 to discard packets to any destinations in 18.1.0.0/16 that do not have a more specific route.

5 FLOODING ATTACKS USING PERSISTENT FORWARDING LOOPS

In this section, we analyze the impact on the bandwidth consumption of the links in persistent forwarding loops and the effort that an attacker requires in order to launch such flooding attacks.

When a packet is sent to a shadowed address, it will fall into the persistent forwarding loop. It traverses the links in the loop and will be dropped only when its TTL expires. Therefore such a packet may traverse the links in the loop multiple times before being dropped and will consume more bandwidth. We define *traffic amplification factor* as the average number of times that a packet traverses a link in a persistent forwarding loop. Typically, a packet has a TTL value with 64 when created at its origin. From our measurement, we find that persistent forwarding loops occur on average $14 \sim 15$ hops away from the source. Without losing generality, we consider that a packet traverses about 14 routers to fall into persistent forwarding loops. The persistent forwarding loops typically have a length of 2 as shown in Section 4.2.2. With this statistics, the traffic amplification factor can be estimated to be $\frac{64-14}{2} = 25$. It means that a packet will traverse the links in forwarding loops 25 times of what is expected. So the persistent forwarding loops can induce much more traffic than expected.

Due to the amplification on the traffic by persistent forwarding loops, attackers are expected to take much less effort to launch flooding attacks on imperiled addresses. For example in Fig. 1, if the available bandwidth for the link L_{bc} is 50Mbps, and traffic amplification factor is 25, then an attacker needs to send traffic at the rate of 2Mbps to flood L_{bc}. If an attacker has compromised 100 computers in the Internet and launches such an attack, the average traffic rate on each machine is only 20Kbps. Such a rate can be easily performed by most users and be hard to detect from the source.

6 SUMMARY

In this paper we investigate the vulnerability on flooding attacks by exploiting persistent forwarding loops. We emphasize that such vulnerability can be exploited from various locations, and can severely affect the Internet connectivity to a significant number of network addresses. These findings suggest that this vulnerability could be a critical threat to the Internet security. In our future work, we plan to study the causes of persistent forwarding loops and how to eliminate these hidden troubles.

7 ACKNOWLEDGMENTS

The authors would like to thank our shepherd Jennifer Rexford for helpful suggestions and anonymous reviewers for valuable comments on the paper. This work was supported in part by NSF grant ANI-0208116, ANI-0085848, and the Alfred P. Sloan fellowship.

References

[1] FRANCOIS, P., AND BONAVENTURE, O. Avoiding Transient Loops during IGP Convergence in IP Networks. In *Proc. IEEE INFOCOM* (March 2005).

[2] HALABI, B. *Internet Routing Architectures*. Cisco Press, 1997.

[3] HENGARTNER, U., MOON, S., MORTIER, R., AND DIOT, C. Detection and Analysis of Routing Loops in Packet Traces. In *ACM Sigcomm Internet Measurement Workshop* (November 2002).

[4] HTTP://WWW.ARIN.NET/WHOIS/ARINWHOIS.HTML.

[5] INTERNET PROTOCOL V4 ADDRESS SPACE. http://www.iana.org/assignments/ipv4-address-space.

[6] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Understanding BGP Misconfiguration. In *Proc. ACM SIGCOMM* (August 2002).

[7] MAO, Z. M., REXFORD, J., WANG, J., AND KATZ, R. Towards an Accurate AS-Level Traceroute Tool. In *Proc. ACM SIGCOMM* (August 2003).

[8] PAXSON, V. End-to-End Routing Behavior in the Internet. In *IEEE/ACM Trans. Networking* (October 1997).

[9] PLANETLAB. http://www.planet-lab.org/.

[10] ROUTE VIEWS PROJECT. http://www.antc.uoregon.edu/route-views/.

[11] SRIDHARAN, A., MOON, S., AND DIOT, C. On The Correlation Between Route Dynamics and Routing Loops. In *Proc. ACM Sigcomm Internet Measurement Conference* (October 2003).

[12] ZHANG, M., ZHANG, C., PAI, V., PETERSON, L., AND WANG, R. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *6th Symposium on Operating Systems Design and Implementation (OSDI'04)* (December 2004).

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

Member Benefits

- Free subscription to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and Open Source, book and software reviews, summaries of sessions at USENIX conferences, and Standards Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to *;login:* on the USENIX Web site.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, on the USENIX Web site.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as security, Linux, Internet technologies and systems, operating systems, and Windows—as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.
- Savings on a variety of products, books, software, and periodicals: see *http://www.usenix.org/membership/specialdisc.html* for details.

SAGE

SAGE is a Special Interest Group (SIG) of the USENIX Association. It is organized to advance the status of computer system administration as a profession, establish standards of professional excellence and recognize those who attain them, develop guidelines for improving the technical and managerial capabilities of members of the profession, and promote activities that advance the state of the art or the community.

USENIX & SAGE Thank Their Supporting Members

USENIX Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ AMD ❖ Asian Development Bank ❖
❖ Cambridge Computer Services, Inc. ❖ Delmar Learning ❖
❖ Electronic Frontier Foundation ❖ Eli Research ❖ Hewlett-Packard ❖ IBM ❖ Intel ❖
❖ Interhack ❖ The Measurement Factory ❖ Microsoft Research ❖ NetApp ❖ Oracle ❖ OSDL ❖
❖ Perfect Order ❖ Raytheon ❖ Ripe NCC ❖ Sun Microsystems, Inc. ❖ Splunk ❖ Taos ❖
❖ Tellme Networks ❖ UUNET Technologies, Inc. ❖

SAGE Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ Asian Development Bank ❖ Fotosearch ❖
❖ Microsoft Research ❖ MSB Associates ❖ Raytheon ❖ Splunk ❖ Taos ❖ Tellme Networks ❖

For more information about membership, conferences, or publications,
see *http://www.usenix.org/*
or contact:
USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA
Phone: 510-528-8649 Fax: 510-548-5738 Email: *office@usenix.org*

ISBN 1-931971-37-4